Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Forschungszentrum Informatik
an der Universität Karlsruhe
Haid-und-Neu-Straße 10-14
76131 Karlsruhe

Studienarbeit

Integration eines übergeordneten Entwurfswerkzeugs für eingebettete HW/SW-Systeme

cand. inform.
Andreas Rentschler

Betreuer : Dipl.-Ing. Benjamin Lutz
Forschungsbereich Elektronische Systeme und Mikrosysteme (ESM)
Forschungszentrum Informatik (FZI)

Verantwortlicher Betreuer : Prof. Dr.-Ing. K.-D. Müller-Glaser
Institut für Technik der Informationsverarbeitung
Universität Karlsruhe (TH)

Korreferent : Prof. Dr.-Ing. R. Dillmann
Institut für Technische Informatik
Universität Karlsruhe (TH)

Begonnen am : 21. August 2006
Eingereicht am : 20. Februar 2007

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstige Hilfsmittel sind vollständig angegeben.

Karlsruhe, den 20. Februar 2007

_____

Inhaltsverzeichnis

Erklärung III

Abstract X

1 Einleitung 1
- 1.1 Umfeld . . . 1
- 1.2 Motivation . . . 1
- 1.3 Gliederung . . . 3

2 Konzeption 5
- 2.1 Unmittelbare Anforderungen . . . 5
- 2.2 Existierende Lösungsansätze . . . 7
- 2.3 Wahl der Zielplattform . . . 8

3 Realisierung 12
- 3.1 Eclipse Rich Client Platform . . . 12
- 3.2 Schnittstellen der Plugins . . . 13
  - 3.2.1 Das Plugin de.fzi.esm.ide.run . . . 14
  - 3.2.2 Das Plugin de.fzi.esm.ide.tools . . . 20
  - 3.2.3 Das Plugin de.fzi.esm.ide.wizard . . . 23
  - 3.2.4 Das Plugin org.eclipse.cdt . . . 24
  - 3.2.5 Das Plugin net.sourceforge.veditor . . . 24
  - 3.2.6 Das Plugin com.svcdelivery.xmlauthor . . . 24
- 3.3 Ausgewählte Teile des Programmkonzeptes . . . 27
  - 3.3.1 Werkzeug zur Generierung von Kostenberichten . . . 27
  - 3.3.2 Eigene Projektvorlage . . . 30
  - 3.3.3 Konfigurierung der Standardwerkzeuge . . . 34

4 Evaluierung 39
- 4.1 Rückblick . . . 39
- 4.2 Bewertung erster Erfahrungen . . . 39
- 4.3 Ausblick . . . 41

A Pflichtenheft 43
- A.1 Zielbestimmung . . . 43
  - A.1.1 Musskriterien . . . 43
  - A.1.2 Wunschkriterien . . . 44
  - A.1.3 Abgrenzungskriterien . . . 44

A.2 Produkteinsatz . . . 44
A.2.1 Anwendungsbereiche . . . 44
A.2.2 Zielgruppen . . . 45
A.2.3 Betriebsbedingungen . . . 45
A.3 Produktübersicht . . . 45
A.4 Produktfunktionen . . . 45
A.5 Produktdaten . . . 57
A.6 Produktleistungen . . . 58
A.7 Benutzungsoberfläche . . . 58
A.8 Nichtfunktionale Anforderungen . . . 70
A.9 Technische Produktumgebung . . . 72
A.9.1 Software . . . 73
A.9.2 Hardware . . . 73
A.9.3 Orgware . . . 73
A.9.4 Produkt-Schnittstellen . . . 73
A.10 Spezielle Anforderungen an die Entwicklungsumgebung . . . 74
A.10.1 Software . . . 74
A.10.2 Hardware . . . 74
A.10.3 Orgware . . . 74
A.11 Gliederung in Teilprodukte . . . 75
A.12 Ergänzungen . . . 75

B Installation **76**
B.1 Eclipse einrichten . . . 76
B.2 Deployment des Systems . . . 77

C Kostenbericht **78**

Glossar **91**

Literatur **93**

Abbildungen

1.1 Ausgerichtete und miteinander verschmolzene Punktwolken des Ferrari 550 Maranello (Quelle: `http://www.geomagic.com`) . . . 2
1.2 Kostenentwicklung zur Fehlerbeseitigung von Fehlern . . . 2
1.3 Von ESDE vorgeschlagener Entwicklungsprozess . . . 3

2.2 Eclipse IDE . . . 8
2.3 Visual Studio .NET 2005 . . . 8
2.1 Das *Altera Prozessmodell* in seiner ursprünglichen Form und erweitert um Drittherstellertools, im Vergleich mit dem *ESDE-Prozessmodell*. . . . 9

3.1 Übersicht über die Eclipse-Architektur (in Anlehnung an [DFK+05] *CD-ROM Companion*) . . . 14
3.2 Abhängigkeiten der Plugins des Systems untereinander und Abhängigkeiten von Fremd-Plugins. . . . 25
3.3 Bereitgestellte und angeforderte Schnittstellen der wichtigsten Plugins . . . 26
3.4 Auswahl der Messdaten . . . 27
3.5 Erstellten Bericht anzeigen . . . 27
3.7 Auswahl der Projektvorlage im New-Project-Wizard . . . 30
3.8 Konfigurieren der Variablen einer Projektschablone . . . 30
3.10 Seite 1 des Startup-Wizard . . . 35
3.11 Seite 2 des Startup-Wizard . . . 35
3.6 Vererbungshierarchie bei der Implementierung einer Werkzeugart . . . 36
3.9 Vererbungshierarchie bei der Implementierung einer Projektvorlage . . . 37
3.12 Wizard zur Pflege der Konfigurationen der Standardtools . . . 38

4.1 Das *Left-Shift*-Prinzip von Andrew Farncombe . . . 40

A.1 Übersicht über die Aktoren im System . . . 46
A.2 Geschäftsprozesse des Anwenders . . . 46
A.3 Seite 1 des Startup-Wizard . . . 61
A.4 Seite 2 des Startup-Wizard . . . 61
A.5 Startup-Wizard manuell starten . . . 61
A.6 Seite 1 des New-Project-Wizard . . . 61
A.7 Seite 2 des New-Project-Wizard . . . 62
A.8 Seite 3 des New-Project-Wizard . . . 62
A.9 Seite 4 des New-Project-Wizard . . . 62
A.10 Fortschrittsanzeige des New-Project-Wizard . . . 62
A.11 C/C++-Perspektive . . . 63

A.12 VHDL-Perspektive 63
A.13 XML-Perspektive 65
A.14 Registerkarte 1 einer Kosimulationskonfiguration 65
A.15 Registerkarte 2 einer Kosimulationskonfiguration 65
A.16 Registerkarte 3 einer Kosimulationskonfiguration 66
A.17 Registerkarte 4 einer Kosimulationskonfiguration 66
A.18 Registerkarte 5 einer Kosimulationskonfiguration 66
A.19 Start einer Kosimulationskonfiguration 66
A.20 Registerkarte 1 einer Programmkonfiguration 67
A.21 Registerkarte 2 einer Programmkonfiguration 67
A.22 Registerkarte 3 einer Programmkonfiguration 67
A.23 Registerkarte 4 einer Programmkonfiguration 67
A.24 Einstellung einer Berichtskonfiguration 69
A.25 Schritt 1 beim Erstellen eines Berichts 69
A.26 Schritt 2 beim Erstellen eines Berichts 69
A.27 Erstellten Bericht anzeigen 69
A.28 Einstellung einer Systemspezifikationskonfiguration 71
A.29 Schritt 1 beim Generieren einer Systemspezifikation 71
A.30 Schritt 2 beim Erstellen einer Systemspezifikation 71
A.31 Schritt 3 beim Erstellen einer Systemspezifikation 71
A.32 Geschäftsprozesse des Erweiterers 71

Tabellen

2.1 Vergleich zweier möglicher Zielplattformen 11

3.1 Präprozessorbefehle innerhalb von Dateien einer Projektvorlage 32
3.2 Hilfsfunktionen zur Registrierung von Variablen unterschiedlichen Typs 34

Abstract

Development of embedded systems is a tedious task involving a large number of specialized tools from different vendors. Such tools, as well as artifacts shared by them, have to be handled in a common environment. This pre-diploma thesis (»Studienarbeit«) describes the development process of a meta-tool supporting management of projects among different tools. A meta-integrated-development-environment called *Embedded Systems Development Environment* (ESDE) is going to be designed and implemented by customizing and extending the *Eclipse* framework. Requirements to ESDE include a wizard to generate projects from pre-defined templates, management of projects and their artifacts in a hierarchical structure, easy management of external programs, implementation and integration of tools operating on artifacts (e.g. creating reports from simulation data), and comfortable viewing and editing of C, XML and VHDL code. Easy accessibility to tools for measuring and flexible evaluation of simulation data as well as extensibility is important above all, as it corrects short-comings of similar environments and assures future use in a scientific context.

Keywords:

Meta-Integrated-Development-Environment (Meta-IDE)
Eclipse framework
Rich Client Plattform (RCP)
Hardware/software cosimulation
Embedded systems development process
FPGA synthesis

1 Einleitung

Noch bevor eine genaue Analyse gemacht werden kann, müssen die Zusammenhänge betrachtet werden die zur Aufgabenstellung dieser Studienarbeit geführt haben. Nach Formulierung der Aufgabenstellung folgt eine Übersicht über die Struktur der Arbeit.

1.1 Umfeld

Die Abteilung Embedded Systems and Sensors Engineering (ESS) des Forschungszentrums Informatik (FZI) Karlsruhe arbeitet zusammen mit dem Unternehmen FARO an der Konzeption von Algorithmen für 3D-Long-Range-Laserscanner und der Synthese dieser Algorithmen im Hardware-/Software-Codesign auf FPGA-Bausteinen. 3D-Laserscanner sind Systeme, die mittels Laserstrahl das zeilen- oder rasterartige Überstreichen von umgebenden Oberflächen und Körpern ermöglichen, um diese zu vermessen. Aus den gewonnen Daten kann ein dreidimensionales Abbild (3D-Modell) der Umgebung erzeugt werden, das virtuell begangen werden kann. Durch hohe Präzision und weitreichende Optik sind unterschiedliche Anwendungen möglich. Nicht nur das Erfassen von Fabrikhallen ist denkbar, auch das Konservieren von Tatorten von Verbrechen für eine genaue spätere Analyse des Tathergangs ist ein möglicher Anwendungsbereich. Abbildung 1.1 zeigt ein Punktmodell einer vermessenen Karosserie aus über 40 Millionen Punkten.

Am FZI wird ein werkzeuggestützter Entwurfsprozess erarbeitet, der Konzeption und Synthese von Algorithmen besonders optimal hinsichtlich Kosten, Entwicklungsaufwand und Ergebnis ermöglicht.

1.2 Motivation

Beim klassischen Entwurf eines Systems wird zunächst die unterste Ebene, die Hardware, entwickelt. Anschließend werden, Schritt für Schritt, die abstrakteren Softwareschichten darauf aufgebaut (**Bottom-Up-Entwurf**). Diese sequentielle Vorgehensweise ist beim Entwurf eingebetteter

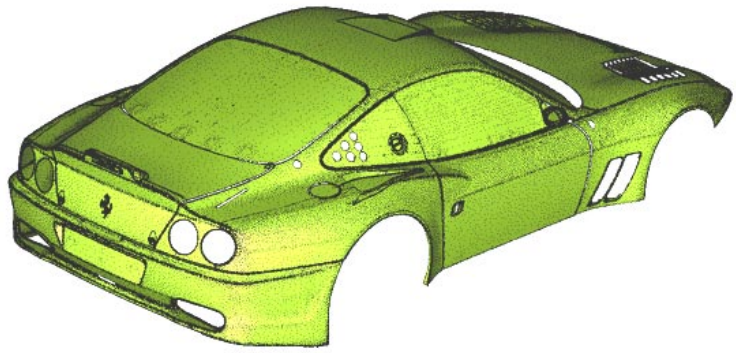


Bild 1.1: Ausgerichtete und miteinander verschmolzene Punktwolken des Ferrari 550 Maranello (Quelle: http://www.geomagic.com)

Systeme selten machbar, da sowohl Hardware als auch Software speziell an die Anforderungen der Zielumgebung anzupassen sind, um möglichst schnell ein möglichst effizientes und kostengünstiges System zu erhalten. Die Entscheidung, welcher Teil der geforderten Funktionalität in Hardware und welcher in Software zu realisieren ist, kann mitunter erst zur Implementierungszeit entschieden werden. Das liegt daran, dass erst durch die Simulation beider Komponenten (**Kosimulation**) eine Aussage gemacht werden kann, ob alle nichtfunktionalen Anforderungen erfüllt werden oder eine Änderung des Entwurfs nötig ist. Die Methodik, Hardware und Software gleichzeitig zu entwerfen, wird **Hardware/Software-Koentwurf** genannt.

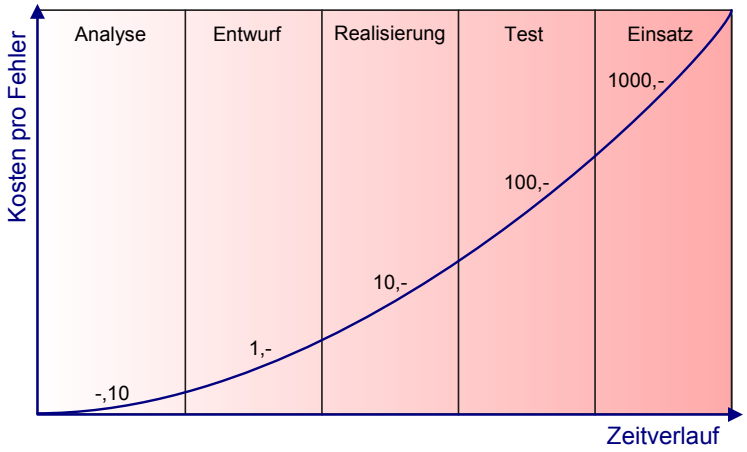


Bild 1.2: Kostenentwicklung zur Fehlerbeseitigung von Fehlern

Der Koentwurf ist eine wichtige Herangehensweise, um ein System von hoher Qualität zu erhalten [MG97]. Dennoch geht diese Art des Entwurfs einher mit hohen Entwicklungskosten, die durch komplexe Entwurfsflüsse und eine große Menge verwendeter Werkzeuge verursacht wird, die einige Aufgaben nicht zufriedenstellend oder gänzlich nicht lösen können. Denn trotz

der Integration etablierter Werkzeuge in den Entwurfsprozess fehlen beispielsweise Funktionen zu einer frühzeitigen Architekturbewertung und Fehlererkennung. Gerade beim kostenintensiven Koentwurf sind diese Funktionen besonders wichtig, da die Beseitigung nicht erkannter Fehler in späteren Entwurfsphasen immer teurer wird. Abbildung 1.2 illustriert die *Zehnerregel*, nach der die Behebungskosten eines Fehlers mit jeder Entwicklungsphase um das Zehnfache zunehmen.

Verbesserung kann an dieser Stelle ein Metawerkzeug schaffen, das eine Schnittstelle zu den einzelnen vorhandenen Werkzeugen darstellt, und das um weitere Werkzeuge ergänzt werden kann, um potentielle Lücken in der Werkzeugunterstützung schließen zu können. Eine Übersicht über den am FZI erarbeiteten Entwicklungsprozess, der eine werkzeuggestützte Abdeckung aller Entwurfsphasen vorsieht, zeigt Abbildung 1.3.

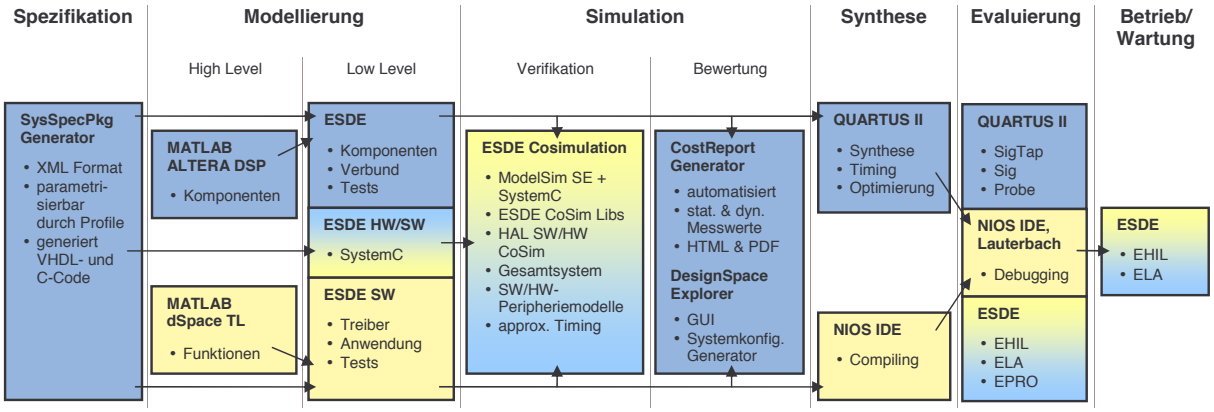


Bild 1.3: Von ESDE vorgeschlagener Entwicklungsprozess

Aufgabe dieser Studienarbeit ist der Entwurf eines adaptierbares Metawerkzeuges, das die komfortable Verwaltung von Projekten und Werkzeugen in einer fensterbasierten Oberfläche ermöglicht und jederzeit neue Funktionen und Werkzeuge in die Oberfläche integrieren kann. Ein neues Werkzeug, das aus der Kosimulation gewonnene Daten automatisiert auswertet und in einem graphischen Bericht visualisiert, soll implementiert und zusammen mit bereits entwickelten Werkzeugen in die Oberfläche integriert werden.

1.3 Gliederung

In Kapitel 2 werden die Anforderungen an das *Meta-Tool* erörtert. Die Unzulänglichkeiten existierender Lösungsansätze werden beschrieben, und es wird entschieden auf welcher Zielplattform das System fußen soll.

Kapitel 3 beschreibt die Grobarchitektur des Systems, die das Ergebnis eines ausführlichen Entwurfsprozesses ist. Drei ausgewählte Teilkonzepte der Architektur werden im Abschnitt 3.3 näher betrachtet.

2 Konzeption

Wie für Softwareprojekte üblich, beginnt jedes Projekt mit einer umfangreichen Analyse der Problemstellung [Bal98]. Zunächst sind methodisch gestützt die Anforderungen an das Programm aufzustellen. Anschließend werden bereits vorhandene Werkzeuge analysiert, sowie eine geeignete Zielplattform gewählt werden.

2.1 Unmittelbare Anforderungen

Um Anforderungen an das Produkt definiert werden können, wird das geplante Einsatzszenario aufgestellt. Im Szenario, das in dieser Studienarbeit betrachtet wird, handelt es sich um einen erweiterten und optimierten Entwurfsprozess für eingebettete HW/SW-Systeme auf der Basis von FPGA-Bausteinen. Wie der Entwurfsfluss mit den aufgezählten Werkzeugen aussehen kann, wurde bereits in Abbildung 1.3 gezeigt. Als Grundlage dieses Entwurfsprozesses dienen alle Tools, die auch im von Altera vorgeschlagenen Standard-Prozess eine Schlüsselrolle spielen:

MentorGraphics ModelSim 6: Dient zur Simulation des in VHDL beschriebenen Hardwaremodells. Ausgelöste Interrupts werden im Simulator abgefangen und mittels der Schnittstelle Foreign Language Interface (FLI) an die Software weitergeleitet.

Altera Quartus II: Mit diesem Werkzeug wird die spätere Synthese des Hardwaremodells durchgeführt und der FPGA programmiert. Im enthaltenen *SoPC Builder* werden alle Komponenten mit dem Altera Bus verdrahtet.

Altera NIOS II IDE: Die Entwicklungsumgebung für den weich konfigurierbaren Mikroprozessor (Soft-Core SoPC) Altera NIOS der zweiten Generation. Sie erlaubt FPGA-Programmierern die Wahl eines Kerns (schnell, ökonomisch oder standard) sowie das Konfigurieren desselben. Eigene Befehle und Peripherie können hinzugefügt werden. Die Oberfläche der integrierte Entwicklungsumgebung (IDE) basiert auf *Eclipse*.

Für die von ESDE vorgeschlagenen Ergänzungen des Standardprozesses sind weitere Anwendungen von Drittherstellern vorgesehen:

Microsoft Visual Studio 6: In der IDE von Microsoft kann die Software des Systems in SystemC entwickelt werden und später im *Debugger* ausgeführt werden. Werden in ModelSim Interrupts ausgelöst, kann so die Ausführung entsprechenden Programmcodes Schritt für Schritt verfolgt werden. Die Verbindung zwischen Hardware und Software basiert auf der ModelSim-Schnittstelle Foreign Language Interface (FLI).

MATLAB : Mit MATLAB lassen sich mathematische Probleme auf numerische Art lösen. Mit Funktionsbibliotheken für die Signalverarbeitung ergänzt, können die später zu modellierenden Algorithmen entworfen und getestet werden.

Ferner sieht ESDE eigene Tools zur Unterstützung der Spezifikations- wie auch der Evaluierungsphase vor:

ELA-View: ELA-View ist ein eigenes Tool zur Visualisierung von Daten, die das zu entwickelnde eingebettete System generiert. Das Tool liegt als ausführbare Datei vor.

System Specification Generator: Mit diesem Hilfswerkzeug können die Eckdaten der Zielplattform spezifiziert werden und anschließend in Form von Konstanten nach VHDL- und SystemC-Code exportiert werden.

Cost Report Generator: Hierbei handelt es sich um ein im Rahmen dieser Studienarbeit noch zu entwickelndes Werkzeug. Sein Zweck ist das Verarbeiten von Messwerten aus der Simulation zur graphischen Ausgabe in einem Kostenbericht.

Design Space Explorer: Dieses Werkzeug wird derzeit innerhalb des Instituts noch entwickelt. Ziel ist die graphische Darstellung der erstellten Software- und Hardwarearchitektur zur einfacheren Bewertung.

Mit der Software sollen die nachfolgend genannten Aufgaben durchführbar sein. Die einzelnen Anwendungsfälle orientieren sich an den im Beispielszenario genannten Werkzeugen.

Gemeinsame Projektverwaltung: Die Verwaltung der Artefakte eines Systementwurfs soll – als Projekt zusammengefasst – zu jedem Zeitpunkt in der Entwicklung möglich sein. Es sollen auch mehrere Projekte gleichzeitig verwaltet werden können. Eine übersichtliche hierarchische Gliederung in Unterordner soll machbar sein.

Grafische integrierte Entwicklungsumgebung (IDE): Eine eingängige, möglichst fensterbasierte Oberfläche soll als Frontend für die Verwaltung der Projekte dienen. Artefakte sollen formatgerecht angezeigt und editiert werden können (Wie für moderne Texteditoren üblich mit *Syntaxhighlighting*). Im Szenario sind dies hauptsächlich die Formate C, VHDL und XML.

Template Wizard: Ein Vorlagengenerator soll die Erstellung von beispielhaften Projekten durchführen. Der Generator soll leicht um eigene Vorlagen erweiterbar sein.

Werkzeuganbindung: Der schnelle Aufruf beliebiger externer Programme soll möglich sein. Artefakte sollen mit beliebigen externen Programmen bequem öffenbar sein.

Werkzeugverbindung: Konverter sollen Artefakte für die Weiterverarbeitung in andere Formate umwandeln.

Auswertung: Bei der Simulation gemessene Daten sollen in Form eines Berichts visualisiert werden, um Aufschlüsse über die Kosten einzelner Module zu geben. Eine flexibel anpassbare Auswertung hilft maßgeblich dabei, Defizite im Entwurf frühzeitig zu erkennen und dadurch unnötige Kosten zu vermeiden. Im Szenario werden in der Simulation Kosten berechnet und in einem XML-Dokument zusammengefasst, anschließend werden die Daten grafisch aufbereitet.

Aus diesen Anforderungen werden später feingranularere Geschäftsprozesse abgeleitet, die die Grundlage für die formal spezifizierten, funktionalen wie auch nichtfunktionalen Anforderungen des Pflichtenheftes bilden. Das Pflichtenheft kann im Anhang A nachgelesen werden. Das Pflichtenheft fasst alle Anforderungen an das zu entwerfende Softwareprodukt zusammen, es liefert die vertragliche Grundlage für alle späteren Entwurfs- sowie Implementierungsentscheidungen, und liefert eine essentielle Grundlage für das Benutzerhandbuch des Systems.

2.2 Existierende Lösungsansätze

Vor der Durchführung eines Softwareprojektes ist es unabdingbar, die derzeitige Marktsituation gut zu analysieren, um Rückschlüsse ziehen zu können, ob es sinnvoller ist, ein bereits existierendes System zu verwenden oder anzupassen.

Ähnliche Werkzeuge existieren bereits, so zum Beispiel *Impulse CoDeveloper Universal* von TRIAS Mikroelektronik [TRI07]. Diese Anwendung begleitet den Entwurfsprozess von der Importierung von ANSI-C-Code bis zur Synthetisierung auf FPGA-Plattformen der wichtigsten Hersteller. Insbesondere unterstützt es den Entwickler bei der Architekturbewertung, indem es Modellsimulationen mit anschließender Auswertung der Simulationsdaten durchführt. Dadurch lassen sich die Performanz betreffende Flaschenhälse frühzeitig erkennen.

Bei diesem, aber auch anderen Tools zeigt sich die fatalste Unzulänglichkeit vor allem in der schlecht änderbaren und nicht erweiterbaren Oberfläche. Gerade die Messdatenerhebung und Auswertung der Simulationsdaten ist nicht ausreichend anpassbar.

Eine andere Alternative bietet das Werkzeug *Universe COVER* von Adveda [SVC07], das als Ergänzung der Altera-Werkzeuge hinzugezogen werden kann. Abbildung 2.1 veranschaulicht, welche Aufwands- und Kostenersparnis hinter der Anwendung des ESDE-Prozessmodells liegen kann: Im direkten Vergleich mit Alteras Standardprozessmodell, das sich nur der Werkzeuge von Altera bedient, ist ca. 30-40% weniger Aufwand nötig, damit fallen ca. 40-60% weniger Kosten an; im Vergleich dazu spart das um *MATLAB* und *Universe COVER* ergänzte Altera-Prozessmodell nur geschätzte 10-20% an Aufwand und lediglich 20-30% an Kosten gegenüber dem Standardprozessmodell ein [LK07].

Der Vergleich zeigt, dass das geschätzte Potential des ESDE-Prozessmodells größer ist als das seiner Konkurrenten. Nicht nur die Qualität der Tools, sondern vor Allem die Qualität der zu Grunde liegenden Prozessmodelle sind für Aufwand und Kosten verantwortlich zu machen. Eine konsequente Werkzeugunterstützung in allen Entwurfsphasen hilft, Aufwand und Kosten zu reduzieren. Eine Oberfläche in der Form eines Metawerkzeugs, die sämtliche in den einzelnen Phasen verwendeten Werkzeuge verwaltet, ist nicht auffindbar.

2.3 Wahl der Zielplattform

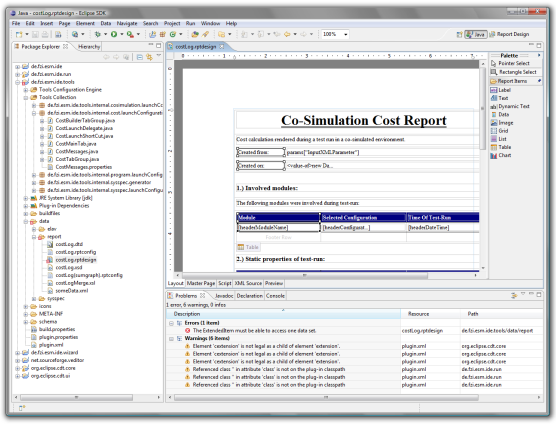


Bild 2.2: Eclipse IDE

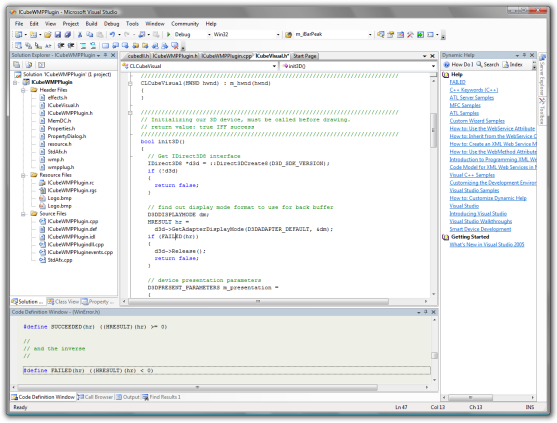


Bild 2.3: Visual Studio .NET 2005

Die Wahl einer geeigneten Zielplattform ist de fakto der Entwurfsphase zugehörig. Dennoch sollte die Entscheidung möglichst noch vor der Entwurfsphase gefällt werden, da sie alle anderen Entwurfsentscheidungen maßgeblich beeinflusst. Im hiesigen Fall kommen zwei unterschiedliche Plattformen in Frage, zum Einen ist dies die vom Unternehmen IBM ins Leben gerufene Eclipse-Plattform, zum Anderen Microsoft Visual Studio. Eine Gegenüberstellung der beiden

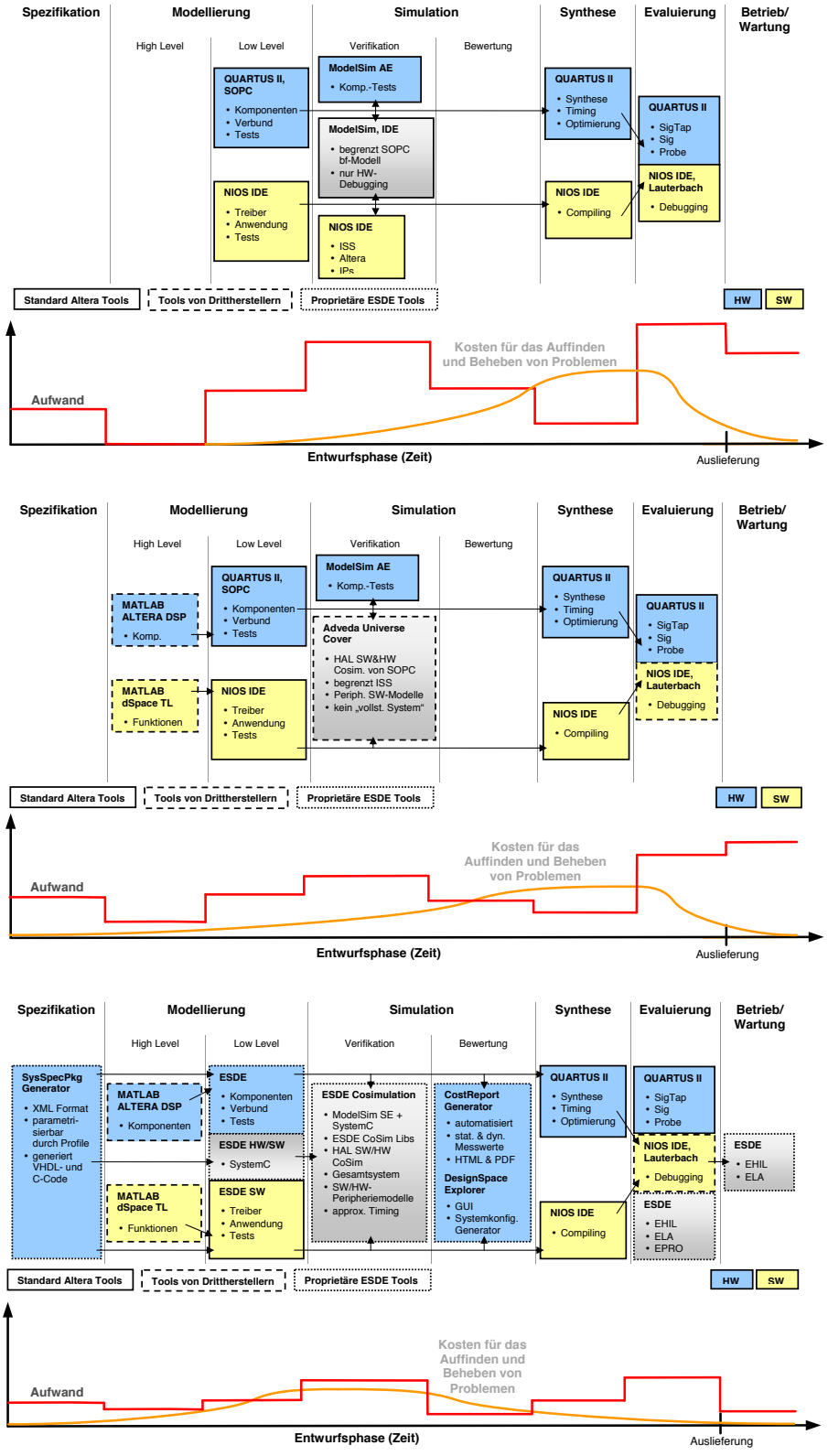


Bild 2.1: Das *Altera Prozessmodell* in seiner ursprünglichen Form und erweitert um Drittherstellertools, im Vergleich mit dem *ESDE-Prozessmodell*.

Plattformen in der Tabelle 2.1 zeigt eine klare Tendenz in Richtung Eclipse auf. Gründe die für Eclipse bzw. gegen Visual Studio sprechen:

1. Die Kombination einer klar strukturierten Komponentenarchitektur gepaart mit vollständig verfügbarem Quellcode aller existierender Komponenten erlaubt die einfachere **Erweiterbarkeit**. Komponenten deren Funktionalität nicht benötigt werden können weggelassen werden. Komponenten, deren Funktionalität in anderer Form zur Verfügung stehen muss, können durch alternative Implementierungen ausgetauscht oder modifiziert werden.

2. Die **Investition** in eine kostenlose, in vollständigem Quellcode vorliegende Plattform ist weniger risikoträchtig als die Investition in eine kommerzielle Plattform, deren Erweiterungsschnittstellen sich in kommenden Versionen grundlegend ändern können. Fehlerhafte Eclipsekomponenten können jederzeit durch neuere Versionen ausgetauscht werden oder aber selbst verbessert werden, während bei einer kommerziellen Plattform auf Bugfixes bzw. Servicepacks seitens des Herstellers abgewartet werden muss.

3. Eine deutlich bessere Verfügbarkeit von **Literatur** für die Eclipseplattform ist festzustellen. Selbst wenn die vorhandene Literatur nicht ausreichen sollte, kann der ausreichend dokumentierte Quellcode der Eclipseplattform zu Rate gezogen werden. Für Visual Studio existiert wenig gedruckte Literatur, neben dem Hilfesystem der SDK müsste auf das Wissen erfahrener Visual-Studio-Erweiterer auf dem eigens hergerichteten Webportal zurückgegriffen werden.

4. Die Eclipseplattform ist für viele **Betriebssysteme** verfügbar. Dieser Punkt ist weniger ausschlaggebend, da viele Tools, die in die ESDE-Entwicklungsumgebung zu integrieren sind, ebenfalls nur für Microsofts Betriebssystem zur Verfügung stehen.

Tabelle 2.1: Vergleich zweier möglicher Zielplattformen

Feature	**Plattform**	
	Visual Studio ab Version 7	**Eclipse ab Version 2.1**
Erweiterbarkeit	umfangreich erweiterbar	vollständig erweiter-/änderbar
– Realisierung	Addins (COM-Komponenten in C++ oder einer .Net-Sprache), Makros (Befehlsfolgen in VB.Net), Templates (Projektvorlagen)	Plugins (OSGi-Komponenten), keine Makros, Vorlagen für Plugins in der JDT möglich
– Schnittstelle	statisch: über fest vorgegebenes Framework erweiterbar	dynamisch: Rich-Client-Platform (»Everything is a plug-in.«)
Verfügbarkeit	closed-source	open-source
– IDE	kommerziell	kostenlos (Eclipse Public License)
– Plattform	Microsoft Windows	Microsoft Windows sowie 10 weitere Plattformen
Erweiterungen	meist kommerziell	meist kostenlos (Eclipse Public License)
– Literatur	lediglich ein Buch [JSY03] offizielle Webseite [Mic07b] und Webportal [Mic07a]	viel Literatur (z.B. [GB05]) offizielle Webseite [Ecl07], viele weitere siehe [Deu07a]
– Off-The-Shelf	C-Editor, kein VHDL-Editor, eigene Projektvorlagen	C-Editor, VHDL-Editor, Projektvorlagen nur für Java

3 Realisierung

Als Plattform des zu entwerfenden Systems wurde die Eclipse-Plattform ausgewählt. Daher wird zunächst auf die Grundzüge der Eclipse-Plattform eingegangen. Die Grobarchitektur des Systems wird in Abschnitt 3.2 besprochen. Im Abschnitt 3.3 wird die innere Struktur interessanter Teile beleuchtet, hierfür ausschlaggebende Entwurfsentscheidungen werden erläutert. Eine vollständige Erklärung der Architektur auf Klassenebene kann nicht für alle Plugins erfolgen, da dies zu viele Seiten füllen würde. Beim Studieren der Schnittstellen der einzelnen Komponenten kann recht schnell in Erfahrung gebracht werden, an welchen Punkten die Eclipse-API erweitert wird und welche Klassen die Erweiterungsschnittstellen implementieren. Es empfiehlt sich, von diesen Klassen als Einstiegspunkt ausgehend den Programmcode zu durchwandern, um einen groben Überblick über die Architektur zu bekommen.

3.1 Eclipse Rich Client Platform

Die Eclipse-Plattform realisiert als *Rich-Client-Platform* einen kleinen Server und lagert möglichst viel Funktionalität auf die Clients aus. Eine Clientkomponente wird im zugrundeliegenden OSGi-Kontext *Bundle*, im Eclipse-Kontext *Plugin* genannt. Die *Provides-Schnittstelle* eines Plugins dient dazu, anderen Plugins einschließlich sich selbst Funktionalität anzubieten, in der Eclipse-Terminologie *Erweiterungspunkte* genannt. Macht ein Plugin Gebrauch von angebotener Funktionalität, muss dies in seiner *Requires-Schnittstelle* festgelegt werden und wird in der Eclipse-Terminologie *Erweiterung* genannt. Die grobgranularste Strukturierung des Systems geschieht also bei der Aufteilung der Funktionalität auf Komponenten entsprechend der Entwurfsregel »separation of concerns«, der Trennung unterschiedlicher Belange.

Damit der Plattformlader eine Komponente als Plugin erkennt, muss es zum Startzeitpunkt im Unterordner `plugins` in einem Ordner oder `jar`-Archiv mit seinem eindeutigen Namen abgelegt sein und einer festgelegten Struktur genügen. Dazu gehört eine dort abgelegte gültige Manifest-Datei `plugin.xml`, in der folgende Informationen enthalten sind:

- Abhängigkeiten zu anderen Plugins (*requires*) werden definiert.

- Klassen, die für andere Plugins sichtbar sind und extern verwendet werden können (*exports*) müssen explizit genannt werden.
- Zur Verfügung gestellte Erweiterungspunkte (*extension-points*) werden ebenfalls im Manifest definiert, und zwar in Form eines Verweises auf die entsprechende Schema-Datei. Es handelt sich um ein gewöhnliches XML-Schema mit Eclipse-Erweiterungen, die Endung ist daher nicht `.xsd`, sondern `.exsd`. Das Schema definiert wie die XML-Syntax im Manifest auszusehen hat. Die Eclipse-Erweiterungen erlauben die Annotierung mit Erklärungen, aus denen automatisch eine Dokumentation generiert werden kann.
- Alle Erweiterungen (*extensions*) werden aufgezählt. Der die Erweiterung definierende Abschnitt muss dem Schema entsprechen, das der jeweilige Erweiterungspunkt festgelegt hat. Manche Erweiterungspunkte verlangen, dass eine Klasse angegeben werden muss, die eine vereinbarte Schnittstelle implementiert, manche Erweiterungspunkte wollen lediglich parametrisiert werden.

Die Eclipse-Plattform an sich besteht aus einer Reihe von Pflichtkomponenten, durch Zusatzkomponenten kann Funktionalität hinzugefügt werden. Erst diese optionalen Komponenten machen aus der Plattform eine neutrale Arbeitsumgebung zur Verwaltung von Ressourcen in Projekten, sowie mit unterschiedlichen Sichten auf die Ressourcen und Editoren für die Ressourcen.

Mit der Absicht, mit der Eclipse-Plattform nicht nur gewöhnliche Javaanwendungen, sondern auch für die Plattform selbst bestimmte, sie erweiternde Plugins in Java schreiben zu können, wurden zwei weitere Projekte ins Leben gerufen: Die Java Development Tools (JDT) erweitern die Eclipse-Plattform zu einer Entwicklungsumgebung für Java-Projekte, darauf aufbauend ermöglicht die Plugin Development Environment (PDE) die Entwicklung von Eclipse-Plugins als eine besondere Ausprägung eines Java-Projektes. Abbildung 3.1 verdeutlicht die Zusammenhänge.

3.2 Schnittstellen der Plugins

Bereits bei der Ermittlung der Geschäftsprozesse konnte eine Einteilung nach den Zuständigkeiten erfolgen (Abbildung A.2). Weiterhin ist es schon hier sinnvoll, sich Gedanken zu machen, welche Funktionalität bereits in vorhandenen Plugins zur Verfügung gestellt wird, getreu der von Erich Gamma und Kent Beck empfohlenen Nachahmungsregel, bei der Entwicklung damit zu beginnen, »die Struktur eines ähnlichen Plugins zu kopieren« (*Monkey See/Monkey Do Rule* [GB05, p.57]). Die Einteilung sieht die in den nachfolgenden Unterkapiteln dargestellten

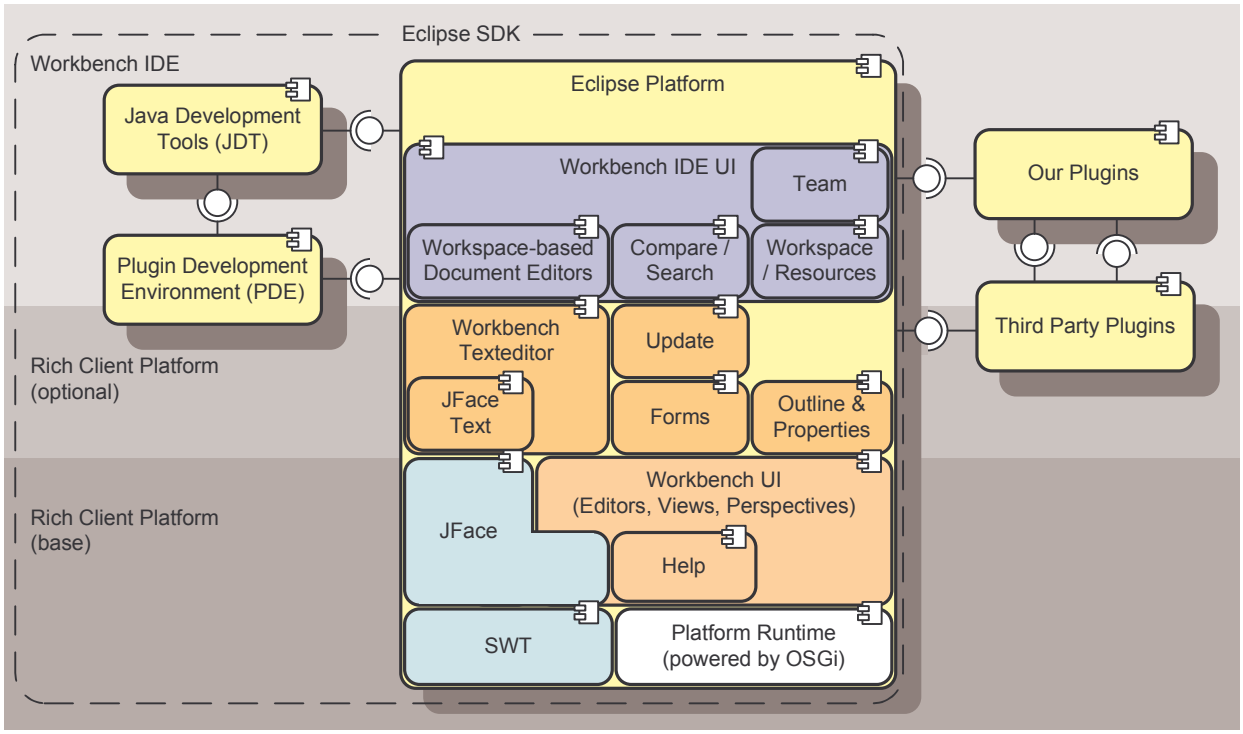


Bild 3.1: Übersicht über die Eclipse-Architektur (in Anlehnung an [DFK+05] *CD-ROM Companion*)

Komponenten vor, eine Übersicht über die Schnittstellen neu entworfener Komponenten ist im Komponentendiagramm in Abbildung 3.3 dargestellt. Einen gröberen Überblick über das System vermittelt Abbildung 3.2, das Diagramm zeigt auf, welche der vom System implementierten Plugins welche Plugins der Eclipse-Plattform benötigen.

3.2.1 Das Plugin de.fzi.esm.ide.run

Die Komponente stellt Funktionalität zur Verwaltung von Konfigurationen unterschiedlicher Typen bereit, welche wiederum in unterschiedliche Kategorien eingeteilt werden können. Diese Funktionalität wird genutzt, um eine eigene Kategorie von Konfigurationen zu implementieren, die für die Ausführung von Projekten zuständig sind. Der in diesem Plugin implementierte Startkonfigurationstyp dient zum Start der beiden nötigen Anwendungen, die zur Cosimulation, d.h. zur Simulation von Hardwareteil und Softwareteil eines ESDE-Projektes vorgesehen sind. Die Struktur der Komponente orientiert sich an denjenigen Teilen des JDT-Plugins *org.eclipse.debug.ui*, welche die Verwaltung von Konfigurationen ermöglichen und in die Oberfläche integrieren. Die Erweiterung um Startkonfigurationen orientiert sich an Teilen des Plugins

org.eclipse.ui.exernaltools, auch in diesem Plugin wird ein spezieller Konfigurationstyp zur Ausführung externer Prozesse implementiert.

Folgende Erweiterungspunkte werden bereitgestellt:

launchConfigurationTabGroups Hier kann ein Konfigurationstyp definiert werden. Durch eine Implementierung der Schnittstelle `ILaunchConfigurationTabGroup` ist eine Gruppe von Registerkarten mit Einstellungen zu definieren. Der Konfigurationstyp mit den entsprechenden Registerkarten wird dem Einstellungsdialog für Start-/Werkzeugkonfigurationen hinzugefügt. Eine mit dem Konfigurationstyp assoziierte Standardperspektive und eine textuelle Beschreibung können angegeben werden, ferner entscheidet die Kategorie der Konfiguration darüber, in welchem der beiden Einstellungsdialoge der Konfigurationstyp erscheint. Neben der in diesem Plugin definierten Startkategorie, der Konfigurationen des Modus `run` zugewiesen werden, wird in *de.fzi.esm.ide.tools* eine weitere Werkzeugkategorie erzeugt, dieser sind Konfigurationen des Modus `tool` zugewiesen.

launchConfigurationTypeImages Der Erweiterungspunkt ermöglicht, für einen über seine eindeutige ID gekennzeichneten Konfigurationstyp ein Icon anzugeben. Das Icon wird mit Konfigurationsinstanzen in der GUI, insbesondere in den Konfigurationsdialogen und Menüs, in Verbindung gebracht.

launchGroups Um eine Kategorisierung von Konfigurationen in Gruppen vornehmen zu können, müssen Gruppen mit diesem Punkt vorher kreiert werden. Eine Gruppe fasst Konfigurationen eines bestimmten Modus in einem eigenen Dialog zusammen. Neben Name, anzuzeigendem Namen und Icon kann ein Dialogbannersymbol und ein Dialogtitel angegeben werden. Einer Kategorie können mehrere Typen hinzugefügt werden, die im linken Auswahlfeld des Dialogs erscheinen. Später existiert beispielsweise eine Kategorie für Startkonfigurationen, ihr sind Konfigurationstypen des Modus `run` zugeordnet. Der Kategorie von Werkzeugkonfigurationen sind dann Konfigurationstypen des Modus `tools` zugeordnet.

launchShortCuts Der Erweiterungspunkt fügt dem Kontextmenü je nach angegebener Liste unterstützter Modi (zum Beispiel *Run...* und/oder *Apply Tool...*) Menüpunkte hinzu, die abhängig von der derzeitigen Auswahl eine Konfiguration des entsprechenden Typs generieren. Ob der Menüpunkt erscheint oder nicht, kann von der Auswahl und der unterstützten Kategorie abhängig gemacht werden. Ein Bezeichner und ein Icon dürfen spezifiziert werden, ebenso in welchem Unterverzeichnis des Auswahlmenüs der Eintrag erscheint. Von der Schnittstelle `ILaunchShortcut` erbend wird die auswahlabhängige Generierung einer Konfiguration ausprogrammiert.

stringVariablePresentation Es wird die Möglichkeit geschaffen, im Dialog zur Auswahl einer Stringersetzungsvariable einen untergeordneten Dialog zu öffnen, der die Auswahl aus einer Menge gültiger Elemente und damit die Parametrisierung der entsprechenden Variablen vornimmt. Die Dialogklasse muss von `IArgumentSelector` erben.

consoleColorProviders Wie Textausgaben der Konsole bestimmter Prozesstypen eingefärbt werden, legen Implementierungen der Schnittstelle `IConsoleColorProvider` fest, zusammen mit der Angabe des Prozesstypus.

consoleLineTrackers Plugins, welche die Ausgabe bestimmter Prozesstypen abhören möchten, können dies über die Schnittstelle `IConsoleLineTracker` tun.

Folgende Erweiterungspunkte werden für Erweiterungen benötigt:

org.eclipse.debug.core.launchConfigurationComparators Der Erweiterungspunkt zielt darauf ab, dass Ausführungskonfigurationen untereinander auf Gleichheit getestet werden können. Erreicht wird dies, indem für jede Konfigurationsvariable eine Vergleichsoperation implementiert wird.

Diese Stelle wird um einen Test der Variable `ATTR_REFRESH_SCOPE` erweitert, einer Variablen, die angibt, welche Ressourcen nach der Ausführung einer Konfiguration aktualisiert werden sollen.

org.eclipse.debug.core.statusHandlers An diesem Punkt kann eine von `IStatusHandler` abgeleitete Klasse festgelegt werden, sowie ein zu überwachendes Plugin und eine zu überwachende Zustands-/Fehlernummer. Löst das Plugin den Zustand aus, wird eine Klasseninstanz mit der Ereignisbehandlung beauftragt.

Dieser Punkt wird erweitert um auf ein ungültiges Ausführungsverzeichnis zu reagieren, eine Konfiguration dann zu löschen, wenn seine Projekte gelöscht werden, und um vor dem Start einer Konfiguration nachzufragen, ob im Editor geladene Ressourcen zuvor gespeichert werden sollen.

org.eclipse.core.runtime.adapters Hier können Adapterfabriken deklariert werden, das sind Klassen, die einen Typ auf einen anderen abbilden können. Der Erweiterungspunkt vermeidet das unnötige Laden von Plugins, da schon über das Manifest entschieden werden kann, ob der gewünschte Adapter vorliegt und es sich lohnt, das Plugin zu laden.

Erweitert wird, um Fabriken anzubieten, die `org.eclipse.core.resources.IProject`, `org.eclipse.core.resources.IFile` und `org.eclipse.ui.part.FileEditorInput` auf die selbst definierte Schnittstelle `ILaunchable` abbilden. Die Schnittstelle dient nur

dazu, die entsprechenden Objekte als ausführbar zu markieren. Später kann die Ausführbarkeit durch einen einfachen Adaptertest überprüft werden.

org.eclipse.core.runtime.preferences Eine Implementierung der Schnittstelle `AbstractPreferenceInitializer` muss die Funktion `initializeDefaultPreferences` enthalten. Diese Methode wird beim Laden der Standardeinstellungen des Plugins aufgerufen, damit das Plugin die Einstellungen selbst auf die Standardwerte setzen kann.

Über die Erweiterungsstelle wird das Setzen der Einstellungsseiten auf die Standardwerte definiert.

org.eclipse.core.variables.dynamicVariables Am Erweiterungspunkt können Variablen genannt werden, sowie eine entsprechende Ersetzungsfunktion via `IDynamicVariableResolver`, die zum Anwendungszeitpunkt die Variable durch eine Zeichenkette ersetzt.

Das Plugin `de.fzi.esm.ide.run` erstellt die Variablen `workspace_loc`, `project_loc`, `project_path`, `project_name`, `resource_loc`, `resource_path`, `resource_name`, `container_loc`, `container_path`, `container_name`, `string_prompt`, `file_prompt`, `folder_prompt` und `selected_text`.

org.eclipse.core.expressions.propertyTesters Erlaubt das Testen von Objekten auf gewisse Eigenschaften innerhalb eines Manifests in der XML-Ausdruckssprache von Manifestdateien. Durch das frühzeitige Testen von Eigenschaften innerhalb von Manifesten kann unnötiges Laden von Plugins vermieden werden. Die anzugebende Testerklasse muss sich von `PropertyTester` ableiten.

Es wird ein Tester der Eigenschaft `launchable` erstellt, der Klassen auf ihre Ausführbarkeit überprüft. Ein weiterer Tester überprüft Ressourcen auf die Eigenschaften `matchesPattern`, `projectNature`, `matchesContentType`, `isContainer`.

org.eclipse.ui.console.consolePageParticipants Der Konsolenseite können Steuerelemente hinzugefügt werden, eine Implementierung von `IConsolePageParticipant` wird über den Lebenszyklus der Konsolenseite benachrichtigt (Erstellen, Aktivierung und Deaktivierung, Schließen). Benachrichtigung der Klasse (und somit das teuere Laden des Plugins) kann von einem XML-Ausdruck abhängig gemacht werden.

Die Fähigkeit des Erweiterungspunktes wird dahingehend genutzt, dass die Konsolensicht eines in der Ausführung befindlichen Prozesses Steuerelemente erhält, mit denen der Prozess beendet werden kann, die angezeigte Seite geschloseen werden kann, oder alle Seiten beendeter Prozesse geschlossen werden können.

org.eclipse.ui.preferencePages Dem von der Workbench zur Verfügung gestellten Einstellungsdialog können eigene Einstellungsseiten hinzugefügt werden. Neben einem Namen und Kategorieverzeichnis muss eine Implementierung von `IWorkbenchPreferencePage` erfolgen.

Dem Dialog werden neben einer allgemeinen Einstellungsseite *Run* eine Unterseite *Launching* mit Optionen für die Ausführung von Konfigurationen, eine Unterseite *String Substitution* für dynamische Variablen und einer Unterseite *Console* für die Anpassung der Konsole eingerichtet.

org.eclipse.ui.keywords Für den Einstellungsdialog können einer Einstellungsseite eine Menge von Schlüsselwörtern zugeordnet werden. Bei der Eingabe einer Zeichenfolge im Suchfeld links oben im Einstellungsdialog werden nur die Einstellungsseiten angezeigt, deren zugeordnete Schlüsselworte die Zeichenfolge enthalten. Der Name einer Einstellungsseite ist automatisch unter der Menge ihrer Schlüsselworte.

Das Plugin fügt allen Einstellungsseiten, die es angelegt hat, Schlüsselworte hinzu.

org.eclipse.ui.actionSets Mit diesem Erweiterungspunkt werden Menüpunkte den Menüleisten und Icons den Werkzeugleisten hinzugefügt. Neben Angaben über den darzustellenden Namen, der Buttonart, den Icons, einer ID für das Hilfesystem und einem der Gruppierung dienenden Menü- bzw. Werkzeugleistenpfad muss eine Implementierung der Schnittstelle `IWorkbenchWindowActionDelegate` oder – für Pulldownmenüs – `IWorkbenchWindowPulldownDelegate` übergeben werden. Weitere Optionen können der Beschreibung des Erweiterungspunktes entnommen werden. Ein XML-Ausdruck kann angegeben werden, damit schon vor dem Laden des Plugins entschieden wird, ob der Menüpunkt bzw. das Icon aktiviert, deaktiviert oder unsichtbar sein soll.

Erweiterungen fügen der Menüleiste ein Menü *Run* mit den Punkten *Run Last Launched*, *Run History*, *Run As* und *Run...* und der Toolbarleiste ein Starticon hinzu.

org.eclipse.ui.elementFactories Der Erweiterungspunkt dient der Definition von Elementfabriken zur Persistierung und Erstellung von persistierten Objekten des Typs `IAdaptable`. Eine solche Fabrik, die von der Schnittstelle `IElementFactory` erbt, erstellt beim Start der Workbench Objekte aus ihrem persistierten Memento heraus. Im Gegenzug dient eine von `IPersistableElement` implementierte Klasse dazu, beim Beenden der Workbench das Objekt aus dem übergebenen Memento heraus zu speichern.

Der Erweiterungspunkt wird zum Persistieren der Ausführungskonfigurationen und der Konfigurationstypen verwendet.

org.eclipse.ui.commands An dieser Stelle können abstrakte Befehle definiert werden. Die konkrete Aktion ist vom Kontext abhängig. So kann derselbe Befehl im Editor eine andere Aktion auslösen als zum Beispiel im aktiven Navigator.

Definiert werden hier die Befehle *Run...* und *Run Last Launched*.

org.eclipse.ui.bindings Verkettet Tastenkombinationen mit bereits festgelegten Befehlen (siehe *org.eclipse.ui.commands*). Verschiedene Tastenkombinationen können zu Schemata zusammengefasst werden und einer Betriebssystemplattform oder einer Nationalität angehören.

org.eclipse.ui.popupMenus Dieser Erweiterungspunkt hilft bei der Realisierung von Kontextmenüs. Es können Menüpunkte für das Kontextmenü bei einer bestimmten Auswahl von Objekttypen (*Object Contribution*) oder im Kontext einer bestimmten Sicht oder einem bestimmten Editor (*Viewer Contribution*) eingerichtet werden. Die Sichtbarkeit kann abhängig von einer gemeinsamen Oberklasse oder den Namen selektierter Objekte (*Wild Cards*) gemacht werden. Alle anderen Eigenschaften sind konform zu den Eigenschaften gewöhnlicher Menüpunkte, die für den Erweiterungspunkt `org.eclipse.ui.actionSets` angegeben werden können.

Das Kontextmenü wird für eine beliebige Selektion von Objekten um ein Untermenü *Run As* erweitert, das durch `RunContextualLaunchAction` mit kontextabhängigen Menüpunkten gefüllt wird. `RunContextualLaunchAction` erbt daher von `IObjectActionDelegate` und – zur Erstellung der Menüs – von `org.eclipse.jface.action.IMenuCreator`. Außerdem wird der Konsolenansicht ein Kontextmenü mit den Unterpunkten *Remove All Terminated...* und *Preferences...* hinzugefügt.

org.eclipse.ui.themes Eigenschaften, die das Erscheinungsbild der Benutzerschnittstelle betreffen, das heißt zu verwendende Farben und Schriftarten, können hier generiert werden und zu einem Thema gruppiert werden. Die Eigenschaft wird automatisch im Einstellungsdialog auf der Einstellungsseite für Farben und Schriftarten aufgeführt (unter *General – Appearance – Colors & Fonts*).

Erstellt wird das Thema *Run* und diesem wird die Eigenschaft *Console Font* hinzugefügt.

de.fzi.esm.ide.run.launchGroups Der Erweiterungspunkt wird von diesem Plugin zur Verfügung gestellt.

Es wird eine Konfigurationskategorie `de.fzi.esm.ide.run` für Startkonfigurationen erstellt. Der untergeordnete Konfigurationstyp für das Ausführen einer Cosimulation wird im Plugin `de.fzi.esm.ide.tools` dazugefügt.

de.fzi.esm.ide.run.stringVariablePresentations Der Erweiterungspunkt wird von diesem Plugin zur Verfügung gestellt.

Erstellt werden Dialoge für die Variablen `resource_path`, `resource_loc`, `resource_name` und `system_property`.

3.2.2 Das Plugin de.fzi.esm.ide.tools

Die Komponente nutzt die Erweiterungspunkte von *de.fzi.esm.ide.run*, um eine neue Konfigurationskategorie dem System hinzuzufügen: Werkzeugkonfigurationen stellen Konfigurationen dar, die, auf eine Auswahl von Ressourcen innerhalb des Projekts angewandt, die selektierten Ressourcen einlesen und entsprechend Ausgaben erzeugen, z.B. neue Ressourcen. Der Werkzeugkategorie wird ein Konfigurationstyp für Systemspezifikationspakete, ein Typ für die Kostenberichte, und ein Typ für beliebige externe Programme hinzugefügt. Ein Teilpaket *de.fzi.esm.ide.tools.internal.startup* kümmert sich um die Einrichtung von Konfigurationen der Standardwerkzeuge. Das Plugin orientiert sich an der Funktionsweise des JDT-Plugins *org.eclipse.ui.externaltools*, insbesondere hinsichtlich des Konfigurationstyps zur Ausführung externer Programme.

Folgender Erweiterungspunkt wird bereitgestellt:

configurationDuplicationMaps Zur internen Erweiterung für durch das Plugin selbst definierte Toolkonfigurationen gedacht, ermöglicht dieser Erweiterungspunkt die Integration von Werkzeugen in den Buildvorgang eines Projektes. Die Konfigurationen für Kostenberichte, Systemspezifikationen und externe Programme definieren diesen Punkt, damit sind alle internen Werkzeuge automatisiert ausführbar. Die Funktionsweise beruht darauf, dass beim Erstellen einer Konfiguration zusätzlich eine Builderkonfiguration erstellt wird. Der zum Konfigurationstyp funktional identische Builderkonfigurationstyp ist ebenfalls zu registrieren.

Folgende Erweiterungspunkte werden für Erweiterungen benötigt:

org.eclipse.debug.core.launchConfigurationTypes Kreiert einen Konfigurationstypen mit einem Namen und einem oder mehreren möglichen Modi. Eine angegebene Kategorie unterstützt die Zuordnung zur Kategorie und damit zum Dialog. Eine Implementierung der Schnittstelle `ILaunchConfigurationDelegate` muss die Logik zur Ausführung von Konfigurationen dieses Typs beinhalten.

Es wird ein Startkonfigurationstyp zur Ausführung von Cosimulationssitzungen registriert. Werkzeugkonfigurationstypen zur Ausführung von externen Programmen und zur Erstellung von Systemspezifikationspaketen und Kostenberichten werden ebenfalls registriert.

org.eclipse.debug.core.launchModes An dieser Stelle werden Ausführungsmodi definiert, denen anschließend Konfigurationskategorien und -Typen zugeordnet werden. Der Modus entscheidet über die Benennung des Befehls zur Ausführung zugeordneter Konfigurationstypen. Dieser Befehlsname muss angegeben werden. Das Plugin `org.eclipse.debug.core` selbst definiert die standardmäßig verfügbaren Modi `run`, `debug` und `profile`.

Ein weiterer Modus `tools` wird addiert mit den Labels *Run Tool* und *Run Tool As...*.

org.eclipse.core.runtime.preferences Funktion und Parameter des Erweiterungspunktes wurden bereits in Abschnitt 3.2.1 besprochen.

Über die Erweiterungsstelle wird das Setzen der in diesem Plugin definierten Einstellungsseiten auf die Standardwerte definiert.

org.eclipse.core.variables.dynamicVariables Funktion und Parameter des Erweiterungspunktes wurden bereits in Abschnitt 3.2.1 besprochen.

Das Plugin `de.fzi.esm.ide.tools` erstellt die Variablen `build_type`, `build_project` und `system_path`.

org.eclipse.core.resources.builders Ein Builder ist ein inkrementeller Projektersteller, die Aufgabe eines Builders ist das Verarbeiten geänderter Ressourcen eines Projektes. Ein Builder kann für spezielle Projektnaturen definiert werden, im vorliegenden Fall zum Beispiel VHDL-Projekte oder C-Projekte. Ein Builder erbt von `IncrementalProjectBuilder`.

Wann immer Änderungen an den Ressourcen eines Projektes gemacht werden oder das Projekt bereinigt wird, werden die in der Builder-Einstellungsseite des Projektes festgelegten Builderkonfigurationen ausgeführt.

org.eclipse.ui.propertyPages Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Die Klasse `ToolsPreferencePage` erweitert den Einstellungsdialog der Workbench um eine Einstellungsseite für Werkzeuge.

org.eclipse.ui.preferencePages Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Die Klasse `BuilderPropertyPage` erweitert den Eigenschaftsdialog von Projekten um eine Einstellungsseite für Builderkonfigurationen.

org.eclipse.ui.actionSets Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Erweiterungen fügen der Menüleiste im Menü *Run* ein Untermenü *Tools* mit den Punkten *Run Tool As*, *Tools...* und *Organize Favorites...* und der Toolbarleiste ein Toolicon mit identischem Untermenü hinzu.

org.eclipse.ui.editors Fügt der Workbench einen neuen Editor hinzu. Eine ausführliche Erklärung liefern Kapitel 19 des Buches [DFK+05] und Kapitel 24 des Buches [GB05].

Ein Editor wird für PDF-Dateien angemeldet. Das Öffnen der Datei geschieht im Standard-Webbrowser des Betriebssystems, in dem in den Meisten Fällen der Acrobat Reader als ActiveX-Plugin eingebettet ist.

org.eclipse.ui.startup Programmlogik die zum Start der Workbench ausgeführt werden soll, wird an dieser Stelle der Plattform in Form einer von `IStartup` abgeleiteten Klasse übergeben. Die Ausführung des Codes kann über den Einstellungsdialog unter *General – Startup & Shutdown* deaktiviert werden.

Zum Start der Workbench wird überprüft, ob alle zur Überwachung eingestellten Konfigurationstypen innerhalb des Workspace existieren. Falls dem nicht so ist, wird ein Wizarddialog aufgerufen, der den Anwender beim Einrichten der Konfigurationstypen hilft oder die Überwachung abschalten lässt.

org.eclipse.ui.commands Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Es werden die Befehle *Run Last Launched Tool* und *Tools...* definiert.

org.eclipse.ui.popupMenus Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Ein Untermenü *Apply Tool...* wird innerhalb des Kontextmenüs eingerichtet, das der Anwendung von Werkzeugkonfigurationen auf eine Auswahl von Objekten dienen soll. Die Klasse `ToolContextualLaunchAction` füllt das Untermenü mit den Konfigurationen.

de.fzi.esm.ide.run.launchGroups Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Es wird eine Konfigurationskategorie `de.fzi.esm.ide.tools` für Werkzeugkonfigurationen erstellt. Eine weitere Kategorie `de.fzi.esm.ide.tools.builders` soll für jede

Werkzeugkonfiguration eine identische Builderkonfiguration erhalten, mit deren Hilfe Werkzeuge an den Buildprozess eines Projektes angehängt werden können. Die Liste der Builder eines Projektes kann unter dem Eigenschaftsdialog des Projektes (Rechtsklick auf das Projekt im *Package Explorer* – *Properties* – *Builders*) eingesehen und geändert werden. Builderkonfiguration sind als versteckt markiert, damit sie nicht im Konfigurationsdialog aufgeführt werden.

de.fzi.esm.ide.run.launchConfigurationTabGroups Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Versieht alle registrierten Konfigurationstypen mit einem Satz Registerkarten die innerhalb des Konfigurationseinstellungsdialog die Einstellung erlauben.

de.fzi.esm.ide.run.launchConfigurationTypeImages Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Versieht alle registrierten Konfigurationstypen mit einem Icon.

de.fzi.esm.ide.run.launchShortcuts Funktion und Parameter des Erweiterungspunktes wurden bereits im Abschnitt 3.2.1 besprochen.

Fügt dem Kontextmenü Menüpunkte hinzu, die selektionsbezogen eine Werkzeugkonfiguration des Typs *System Specification Package Generator* bzw. *Cost Report Generator* erstellen und ausführen.

de.fzi.esm.ide.tools.configurationDuplicationMaps Der Erweiterungspunkt wird vom Plugin selbst zur Verfügung gestellt. Funktion und Parameter des Erweiterungspunktes wurden bereits besprochen.

Ermöglicht allen registrierten Konfigurationstypen die Verwendung im Buildprozess eines Projektes, indem eine Abbildung vom Konfigurationstyp auf seinen entsprechenden Builderkonfigurationstyp gesetzt wird.

3.2.3 Das Plugin de.fzi.esm.ide.wizard

Für die Erstellung von Projekten aus Vorlagen ist diese Komponente zuständig. Eine ähnliche Aufgabe hat das Plugin *org.eclipse.pde.ui*, es fügt der Oberfläche einen Wizard für die Erstellung neuer Pluginprojekte aus Vorlagen hinzu. Teile von Struktur und Implementierungen können von diesem Plugin kopiert werden.

Folgender Erweiterungspunkt wird bereitgestellt:

projectContent Eine Erweiterung dieses Punktes fügt dem Wizard eine weitere Vorlage hinzu. Neben der Angabe eines repräsentativen Icons und Namens für das Auswahlmenü muss die Schnittstelle `IProjectContentWizard` implementiert werden.

Folgende Erweiterungspunkte werden für Erweiterungen benötigt:

org.eclipse.ui.newWizards An dieser Stelle werden Wizarddialogseiten registriert, die zur Erstellung eines bestimmten Ressourcentyps gedacht sind. Zu einem Wizard kann seine übergeordnete Kategorie in der Wizardliste, ein Icon, sein Name, und nicht zuletzt eine Implementierung abgeleitet von der Schnittstelle `INewWizard` angegeben werden.

de.fzi.esm.ide.wizard.projectContent Dieser Erweiterungspunkt wird vom Plugin selbst angeboten.

Zum Zeitpunkt der Studienarbeit existiert lediglich eine Vorlage *Minimal Cosimulation Project* für ein minimales ESDE-Projekt mit einigen Steuerelementen zur Parametrisierung der Vorlage.

3.2.4 Das Plugin org.eclipse.cdt

Die Komponente erweitert die Plattform um einen vollständigen Editor für C/C++-Quelldateien. Ein Teil der Funktionalität wird vom gleichnamigen Plugin übernommen. Dieses Plugin teilt sich auf in das Plugin *org.eclipse.cdt* für die Kernfunktionalitäts-Ebene sowie das Plugin *org.eclipse.cdt.ui* für die User-Interface-Ebene.

3.2.5 Das Plugin net.sourceforge.veditor

Die Komponente erweitert die Plattform um einen vollständigen Editor für VHDL-Code. Sie kann weitestgehend die gleichnamige, frei verfügbare Komponente[T+07] übernehmen. Anpassungen bezüglich der graphischen Erscheinungsform (Icons, Labels, etc.) müssen noch erfolgen.

3.2.6 Das Plugin com.svcdelivery.xmlauthor

Die Komponente erweitert die Plattform um einen vollständigen Editor für XML-Dateien. Auch diese Komponente kann weitestgehend von der gleichnamigen, frei verfügbaren Komponente [SVC07] übernommen werden. Bezüglich der graphischen Erscheinungsform sind hier ebenfalls Anpassungen nötig.

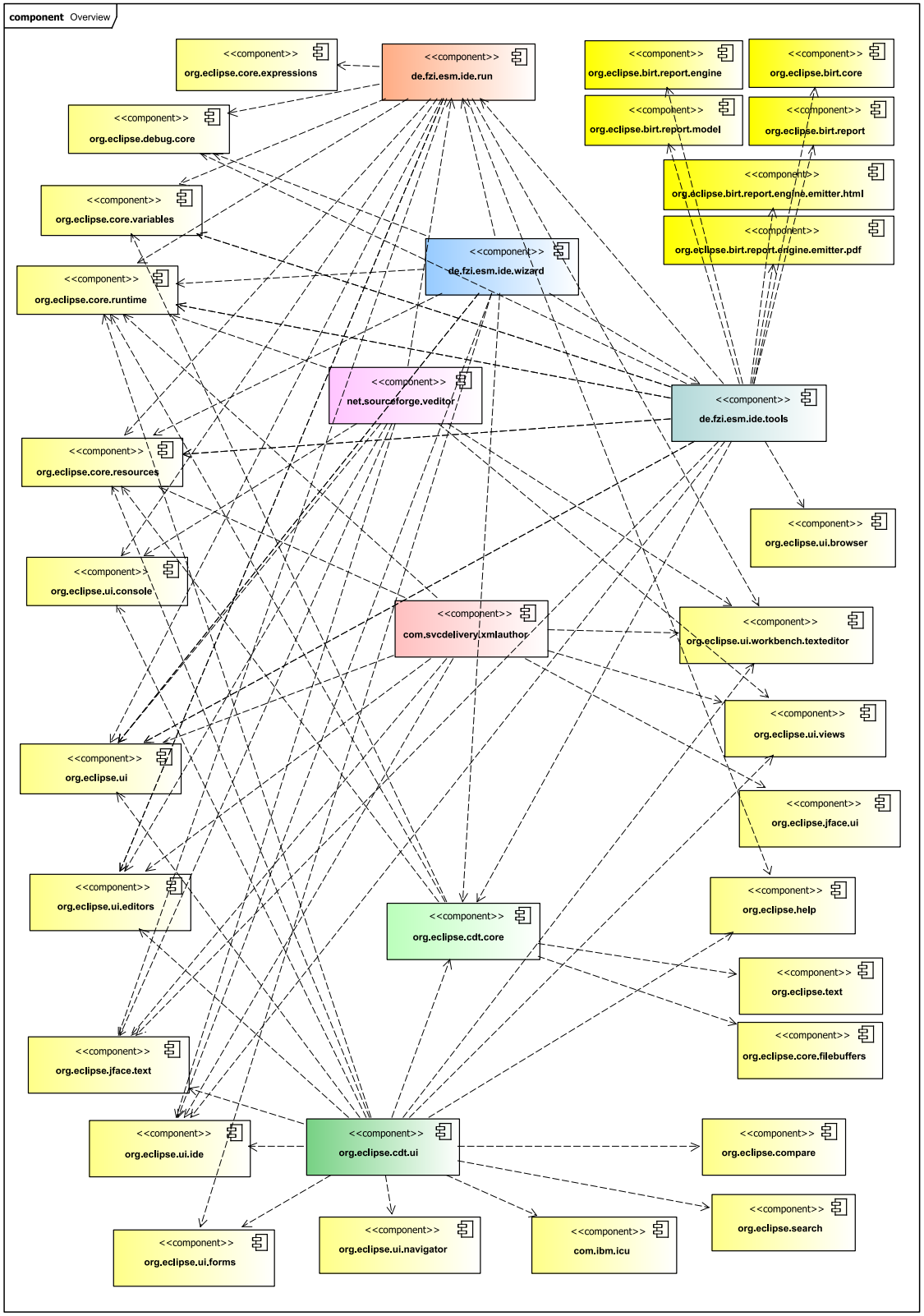


Bild 3.2: Abhängigkeiten der Plugins des Systems untereinander und Abhängigkeiten von Fremd-Plugins.

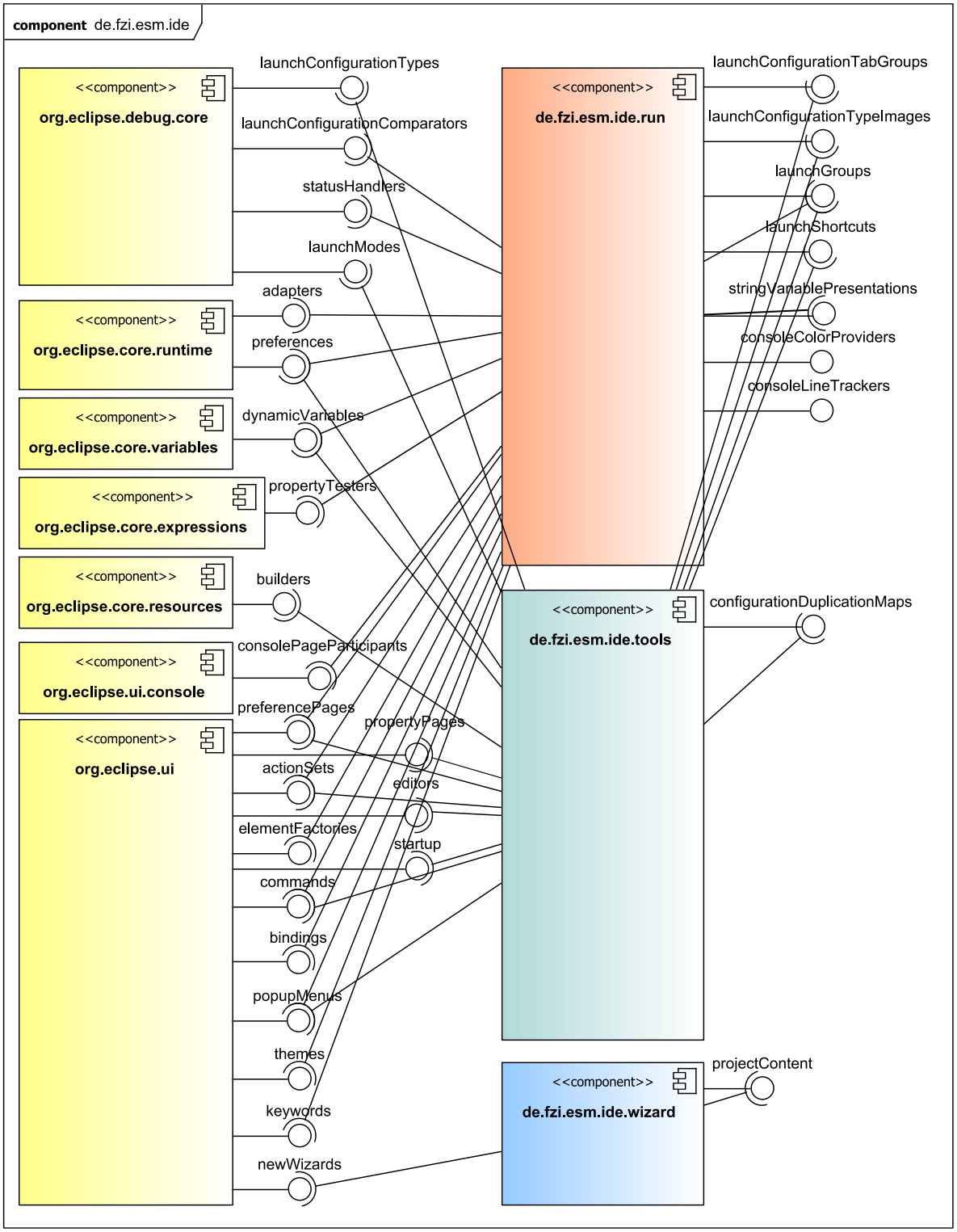


Bild 3.3: Bereitgestellte und angeforderte Schnittstellen der wichtigsten Plugins

3.3 Ausgewählte Teile des Programmkonzeptes

Eine Untersuchung der gesamten Feinarchitektur des Systems wäre zu umfangreich. Darum konzentriert sich dieser Abschnitt auf drei Teile des Architekturkonzeptes.

Zuerst wird der Teil des Systems, der für die Integration von Werkzeugen zuständig ist näher betrachtet, im zweiten Unterabschnitt wird erklärt, wie Projektvorlagen aufgebaut sind und an das System angeschlossen sind. Der dritte und letzte Unterabschnitt befasst sich mit dem Konfigurationssetup.

Diese Bereiche der Architektur sind insbesondere im Hinblick auf die Erweiterung des Systems wichtig.

3.3.1 Werkzeug zur Generierung von Kostenberichten

In Abschnitt 3.2.2 wurde über die Schnittstelle des Plugins `de.fzi.esm.ide.tools` berichtet. In diesem Plugin sind alle Werkzeuge gesammelt, die das System realisiert. Eines dieser Werkzeuge ist für die Auswertung von Simulationsdaten zuständig, seine implementierenden Klassen liegen innerhalb des Pakets `de.fzi.esm.ide.tools.internal.cost.launchConfigurations`. Die Simulationsdaten liegen im XML-Format kodiert vor, wobei jedes VHDL-Modul eine eigene XML-Datei anlegt.

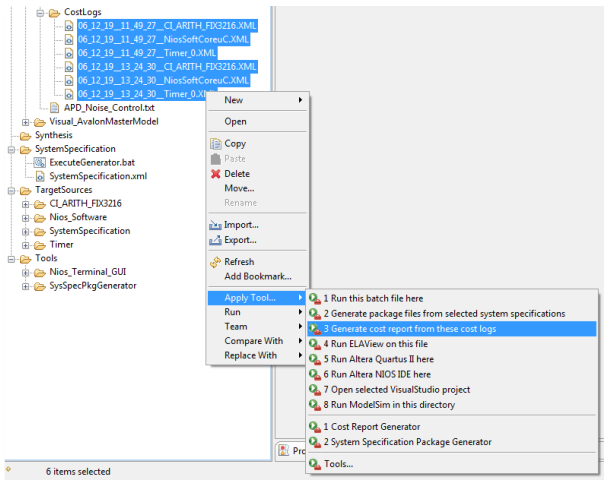


Bild 3.4: Auswahl der Messdaten

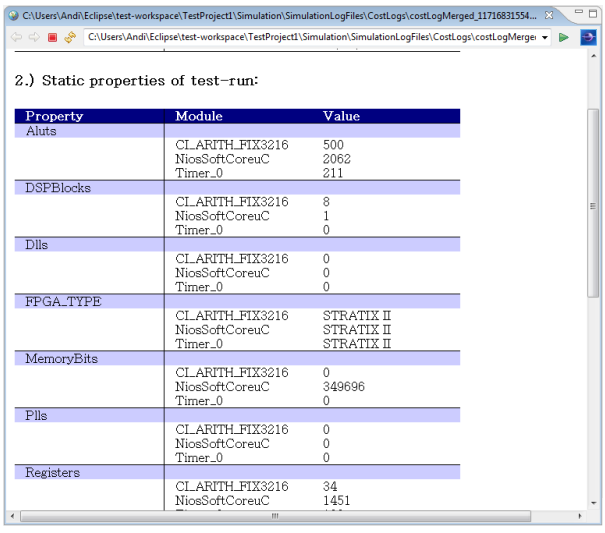


Bild 3.5: Erstellten Bericht anzeigen

Nachdem die Module ausgewählt worden sind (Abbildung 3.4), kumuliert das Werkzeug alle Messdaten intern in einer einzigen Datei, die dann dem Generator übergeben werden. Das Ku-

mulieren der Daten führt die XSL-Transformation `/data/report/costLogMerge.xsl` durch. Listings von Messdatenschema und Kumulationsskript befinden sich im Anhang C).

Als Generator für den Bericht wurde Business Intelligence Reporting Tool (BIRT) gewählt. BIRT ist ein kostenfreies, quelloffenes, auf Eclipse basierendes System, das Lösungen im Bereich der Geschäftsanalyse bereitstellt. Das im Rahmen dieser Studienarbeit erstellte BIRT-Layout `/data/report/costLog.rptdesign` wertet die Daten aus und erstellt je nach Einstellung eine PDF- bzw. eine HTML-Datei mit einem Bilderverzeichnis an der Stelle im Workspace, an der die Messdateien liegen. Eine Einführung in die Funktionsweise von BIRT bietet [PHH06]. In Abbildung 3.5 wird ein generierter Bericht in einer Eclipse-Sicht angezeigt.

Abbildung 3.6 zeigt alle realisierenden Klassen und involvierte Schnittstellen auf. Die Implementierung des Werkzeugs ist von folgenden Erweiterungspunkten abhängig:

org.eclipse.debug.core.launchConfigurationTypes Der Erweiterung wird die Klasse `CostLaunchDelegate` übergeben, an sie wird die Ausführung einer Konfiguration weitergeleitet. Da dem Konfigurationstyp ein weiterer Konfigurationstyp für den Buildprozess zugeordnet ist, der dieselbe Funktionalität während der Buildphase durchzuführen hat, wird auch für dessen Erweiterung die Klasse `CostLaunchDelegate` definiert. Diese Klasse implementiert indirekt die Schnittstelle `ILaunchConfigurationDelegate`. Die Methode `launch(...)` (Listing 3.1) enthält den Code, der die dialoggeführte Auswahl der Dateien realisiert, diese Dateien zu einer Datei zusammenführt, und danach den BIRT-Generator startet.

Listing 3.1: Die Schnittstelle `ILaunchConfigurationDelegate`

```
public interface ILaunchConfigurationDelegate {
    /**
     * Launches the given configuration in the specified mode,
     * contributing targets and/or processes to the given launch
     * object. The launch object has already been registered with
     * the launch manager.
     *
     * @param configuration     the configuration to launch
     * @param mode      the mode in which to launch, in our case either
     *                  ILaunchManager.RUN_MODE or IToolConstants.TOOL_MODE.
     * @param monitor   progress monitor, or null
     * @param launch    the launch object to contribute
     *                  processes and targets to
     * @exception CoreException if launching fails
     */
    public void launch(ILaunchConfiguration configuration, String mode,
            ILaunch launch, IProgressMonitor monitor)
            throws CoreException;
}
```

de.fzi.esm.ide.run.launchConfigurationTabGroups Die Klasse `CostTabGroup` (Listing 3.2), bzw. die Klasse `CostBuilderTabGroup` für den korrespondierenden Builder (Listing 3.3), ist maßgeblich an der Erstellung der Registerkarten beteiligt. Beide Klassen erben die allen Konfigurationsarten gemeinen Implementierungen von der Klasse `AbstractLaunchConfigurationTabGroup`. Auch die Registerkarten für allgemeine Einstellungen von Konfigurationen (`SlimCommonTab`) bzw. Builderkonfigurationen (`ToolsBuilderTab`) können wiederverwendet werden. Einzig und allein die Registerkarte, die toolspezifische Einstellungen ermöglicht, muss vom Tool selbst realisiert werden, im hiesigen Falle ist dies die Klasse `CostMainTab`.

Listing 3.2: Die Klasse `CostTabGroup`

```
public class CostTabGroup extends AbstractLaunchConfigurationTabGroup {
    /**
     * @see de.fzi.esm.ide.run.ILaunchConfigurationTabGroup#createTabs(
     *      de.fzi.esm.ide.run.ILaunchConfigurationDialog, java.lang.String)
     */
    public void createTabs(ILaunchConfigurationDialog dialog, String mode) {
        ILaunchConfigurationTab[] tabs = new ILaunchConfigurationTab[] {
            new CostMainTab(),
            new SlimCommonTab()
        };
        setTabs(tabs);
    }
}
```

Listing 3.3: Die Klasse `CostBuilderTabGroup`

```
public class CostBuilderTabGroup extends AbstractLaunchConfigurationTabGroup {
    /**
     * @see de.fzi.esm.ide.run.ILaunchConfigurationTabGroup#createTabs(
     *      de.fzi.esm.ide.run.ILaunchConfigurationDialog, java.lang.String)
     */
    public void createTabs(ILaunchConfigurationDialog dialog, String mode) {
        ILaunchConfigurationTab[] tabs = new ILaunchConfigurationTab[] {
            new CostMainTab(),
            new ToolsBuilderTab(),
        };
        setTabs(tabs);
    }
}
```

de.fzi.esm.ide.run.launchConfigurationTypeImages Icons für den Konfigurationstyp und den entsprechenden Builderkonfigurationstyp werden an dieser Stelle definiert, um den Wiedererkennungswert in der IDE zu steigern.

de.fzi.esm.ide.tools.configurationDuplicationMaps Das Erstellen einer Konfiguration dieses Typen soll die gleichzeitige Erstellung einer Konfiguration des entsprechenden Typen

für die Buildstufe induzieren. Der Erweiterungspunkt `configurationDuplicationMaps` definiert das Mapping der beiden Konfigurationstypen und erspart damit einige Zeilen Javacode.

de.fzi.esm.ide.run.launchShortcuts Es ist nicht zwingend erforderlich, einen Shortcut zur Ausführung einzurichten. Ein solcher Shortcut erscheint je nach aktiv ausgewählten Ressourcen im Kontextmenü unter dem Menüpunkt *Apply Tool...* und dient dazu, aus dem Kontext der Auswahl eine Ausführungskonfiguration des entsprechenden Typs neu zu generieren. Für das hier besprochene Tool wird die dahinter stehende Funktionalität von der Klasse `CostLaunchShortCut` übernommen. Sie erstellt eine neue Konfiguration, deren einzulesende Messdaten fest aus der zum Erstellungszeitpunkt bestehenden Auswahl besteht, und die den Bericht in beiden vorhandenen Formaten (PDF und HTML) ausgibt.

Es hat sich als sinnvoll erwiesen, alle Textkonstanten in einer eigenen, von der abstrakten Klasse `org.eclipse.osgi.util.NLS` abgeleiteten Klasse zu sammeln, hier `CostMessages`. Dieses Vorgehen unterstützt die Internationalisierung (*National Language Support*). Die Eclipse JDT beinhaltet von Haus aus einen Wizard zur Externalisierung von Zeichenketten.

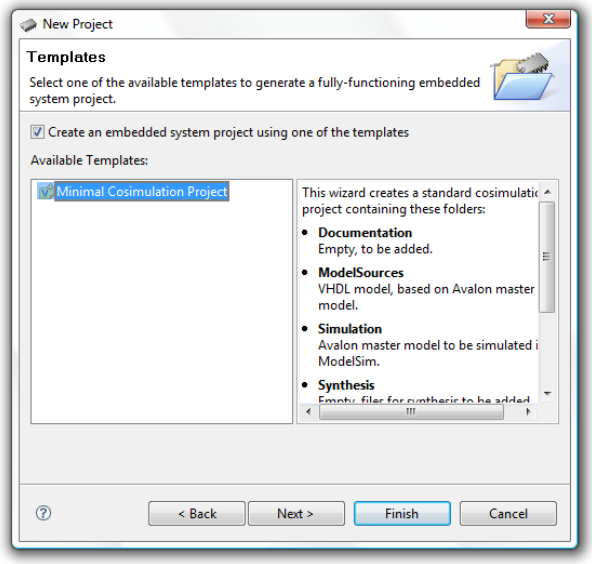


Bild 3.7: Auswahl der Projektvorlage im New-Project-Wizard

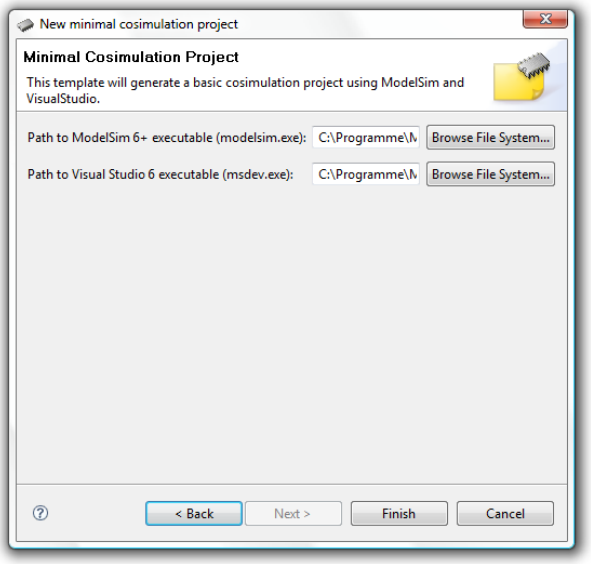


Bild 3.8: Konfigurieren der Variablen einer Projektschablone

3.3.2 Eigene Projektvorlage

Projektvorlagen sind im ESDE-eigenen *NewProject Wizard* aufgeführt (Abbildung 3.7) und erlauben an dieser Stelle das Generieren eines neuen Projektes mit einem fest vorgegebenen Inhalt.

Innerhalb des Wizard-Dialogs kann eine Dialogseite integriert werden, auf der Variablen mit Werten belegt werden können. Beispielsweise dient die Einstellungsseite in Abbildung 3.8 der Konfiguration zweier Variablen. In der Generationsphase schließlich werden alle Verzeichnisse und Ressourcen der Vorlage beim Kopieren in das leer erstellte Projekt mit einem Parser nach Variablen und Ausdrücken durchforstet. Gefundene Variablen werden durch den entsprechend gewählten Wert ersetzt, Ausdrücke werden ausgewertet, sie können beispielsweise das Einfügen eines Textabschnittes von einer boolschen Variablen abhängig machen.

Die Schritte beim Erstellen einer eigenen Projektvorlage lassen sich am Besten an Hand der bereits implementierten Vorlage »Minimal Cosimulation Project« erklären, sie stellen einen programmatischen Einblick in die Feinarchitektur des betroffenen Systemabschnitts dar.

1. Alle Ressourcen des vorhandenen Projekts, das als Vorlage dienen soll, muss zuerst in den Unterordner `/templates/<Projektname>` des Plugins `de.fzi.esm.ide.wizard` kopiert werden. Der Unterordner des Beispielprojekts wurde `"minimalCosimulation-Project"` genannt.

2. Anschließend werden Stellen innerhalb der Ressourcen, die später dynamisch mit Werten belegt werden sollen, mit Hilfe der Templatepräprozessorsprache gekennzeichnet. Im Beispiel musste innerhalb der Datei `/Simulation/Visual_AvalonMasterModel/VISUAL_AVALONMASTERMODEL.dsp` ein absoluter Pfad zur Anwendung *ModelSim* durch die Variable `$modelSimPath$` ersetzt werden. Eine Liste aller verfügbaren Präprozessorbefehle zeigt Tabelle 3.1.

3. Am Erweiterungspunkt `de.fzi.esm.ide.wizard.projectContent` muss die Projektvorlage angemeldet werden. Die Erweiterung implementiert dafür die Schnittstelle `IProjectContentWizard`. Die Erweiterung sollte ebenfalls Icon und Name festlegen, um im Auswahldialog für Projektvorlagen die Vorlage bestmöglich identifizierbar zu machen. Im Beispiel heißt diese, die Erweiterungschnittstelle implementierende Klasse, `MinimalCosimulationNewWizard`.

   Ein großer Teil der Funktionalität von `MinimalCosimulationNewWizard` wird bereits von der abstrakten Klasse `NewProjectTemplateWizard` übernommen, z. B. das Verwalten von aus einem oder mehreren Wizardseiten bestehenden Abschnitten, die durch die Schnittstelle `ITemplateSection` repräsentiert werden. Es wird lediglich ein Abschnitt erstellt, der sich von `IDETemplateSection` ableitet und aus einer einzigen Einstellungsseite besteht, auf der alle Variablen (hier `$modelSimPath$` und `$visualStudioPath$`) vom Anwender mit Werten belegt werden können. Dies geschieht innerhalb der Methode `createOptions()`:

Tabelle 3.1: Präprozessorbefehle innerhalb von Dateien einer Projektvorlage

Präprozessorbefehle

`$templatepreprocessor:$<Zeilenumbruch>`
Anweisung wird durch das $-Zeichen ersetzt

`$templatepreprocessor:<Text ohne Dollar-Zeichen>)$<Zeilenumbruch>`
Anweisung wird ignoriert

`$templatepreprocessor:if(<Ausdruck>)$<Zeilenumbruch>`
if-Befehl: Die nachfolgenden Zeilen bis zum zugehörigen (d.h. auf gleicher Verschachtelungstiefe befindlichen) *else*- oder *endif*-Befehl werden ignoriert
$\Leftrightarrow$ <*Ausdruck*> evaluiert zu *true*.

`$templatepreprocessor:else(<Ausdruck>)$<Zeilenumbruch>`
else-Befehl: Die nachfolgenden Zeilen bis zum zugehörigen *endif*-Befehl werden ignoriert
$\Leftrightarrow$ <*Ausdruck*> des zugehörigen *if*-Befehls evaluierte zu *false*.

`$templatepreprocessor:endif(<Ausdruck>)$<Zeilenumbruch>`
endif-Befehl: Beendet zugehörige *if*-/*else*-Befehlsfolge.

`$<gültigerVariablenname>)$`
Der Inhalt der Variable wird an Stelle der Präprozessoranweisung ausgegeben.

BNF-Grammatik der Nichtterminalsymbole

```
<Ausdruck> ::= <Ausdruck> '&&' <Ausdruck> |
      <Ausdruck> '||' <Ausdruck> | <Ausdruck> '==' <Ausdruck> |
      <Ausdruck> '!=' <Ausdruck> | '(' <Ausdruck> ')' |
      '!' <Ausdruck> | <Wert>
```
BNF-Grammatik für <*Ausdruck*>.

```
<Wert> ::= 'true' | 'false' | '"' (<beliebigesZeichen>)* '"' |
      (<beliebigeZiffer>) | <gültigerVariablenname>
```
BNF-Grammatik für <*Wert*>

`<gültigerVariablenname>`
Name einer für den Präprozessor deklarierten Variable. Die Variable muss ein Steuerelement auf der Projektwizardseite zugewiesen bekommen, das als Eingabemöglichkeit dient. Die Variable `projectName` ist bereits vergeben, sie steht für den Projektnamen.

Listing 3.4: In der durch den Konstruktor aufgerufenen Methode `createOptions()` werden Variablen registriert

```
/**
 * Add options to wizard.
 * Options can be used in code templates and expressions as £optionName£ later on.
 */
private void createOptions() {
    addFolderOption(
        KEY_MODELSIMPATH,
        WizardMessages.MinimalCosimulationTemplate_modelSimPath,
        WizardMessages.MinimalCosimulationTemplate_defaultModelSimPath,
        0);
    addFolderOption(
        KEY_VISUALSTUDIOPATH,
        WizardMessages.MinimalCosimulationTemplate_visualStudioPath,
        WizardMessages.MinimalCosimulationTemplate_defaultVisualStudioPath,
        0);
}
```

Je nach Typ der Variable gibt es neben `addFolderOption(...)` noch andere Hilfsfunktionen, die entsprechende Steuerelemente erzeugen können. Tabelle 3.2 zeigt alle vordefinierten Steuerelemente. Das Validiieren des Inhalts ist durch die Schablonenmethode

```
void validateOptions(TemplateOption source)
```

durchführbar.

Nach der Generierung der Projektressourcen kann die Klasse `MinimalCosimulationNewWizard` weitergehende Arbeiten vornehmen. Hierzu kann sie die Schablonenmethode

```
boolean performFinish(IProject project, IProgressMonitor monitor)
```

implementieren. Im Beispiel wird dem Workspace eine Startkonfiguration hinzugefügt, über die das neu erstellte Projekt gestartet werden kann. Dafür wird ein eigener Thread gestartet, der durch die Klasse `NewMinimalCosimulationCreationOperation` verkörpert wird.

Es gilt zu beachten, dass in der Vererbungshierarchie zwischen `ITemplateSection` und der abstrakten Klasse `IDETemplateSection` in mehreren Stufen verfeinert wird. Zuerst wird die abstrakte Klasse `AbstractTemplateSection` gebildet, das die Schnittstellen `ITemplateSection` und `IVariableProvider` erbt. Damit sind GUI-Elemente der Variablen ebenso wie eine Schnittstelle zur Variablenabfrage vereint. Implementiert werden jedoch an dieser Stelle nur der Generator für das zu erstellende Projekt. Er rekursiert in den Dateibaum der Projektvorlage hinein

Tabelle 3.2: Hilfsfunktionen zur Registrierung von Variablen unterschiedlichen Typs

Methode	Funktion
`addOption(String name, String label, boolean value, int pageIndex)`	ein Häkchen kann gesetzt werden (boolscher Wert)
`addOption(String name, String label, String value, int pageIndex)`	ein einfaches Editierfeld für Zeichenketten
`addFileOption(String name, String label, String value, int pageIndex)`	ein Editierfeld mit zusätzlichem Schalter, mit dem nach einer Datei gesucht werden kann
`addFolderOption(String name, String label, String value, int pageIndex)`	ein Editierfeld mit zusätzlichem Schalter, mit dem nach einem Pfad gesucht werden kann
`addOption(String name, String label, String[][] choices, String value, int pageIndex)`	Radiobuttons lassen die Auswahl aus mehreren Elementen zu
`addBlankField(int height, int pageIndex)`	ein vertikaler Abstandshalter zur optischen Trennung zwischen Steuerelementen

und parst während des Kopiervorgangs nach Befehlen die er ausführt und Variablen die er ersetzt. Die GUI-Repräsentation der Variablen wird erst in der nächsten Verfeinerungsstufe implementiert. Hier wird die Verwaltung der Variablen in Form von Optionen (`ITemplateOption`) eingeführt. Optionen können registriert, initialisiert und abgefragt werden. In der nächsten Spezialisierungsstufe `OptionTemplateSection` wird die Verwaltung der Optionen auf mehreren Wizard-Seiten realisiert. Die letzte Spezialisierung findet durch die Klasse `IDETemplateSection` statt, hier wird der Pfad festgelegt, in dem die Vorlage gesucht wird. Der unterste Verzeichnisname wird hier noch durch die Methode `String getSectionId()` offen gelassen und in der eigentlichen Vorlagenimplementierung als Rückgabewert `"minimalCosimulation"` festgelegt.

3.3.3 Konfigurierung der Standardwerkzeuge

Eine bestimmte Auswahl von Tools gilt als wichtige Grundlage für den Entwurf mit der ESDE. Da für die Anbindung dieser Tools bekannt sein muss, an welcher Stelle im Dateisystem die Anwendung installiert wurde, ist eine Interaktion mit dem Anwender unabdingbar. Konfigurationen werden innerhalb des Workspace erstellt, wird also ein neuer Workspace eingerichtet, existieren zunächst keine Konfigurationen.

Bei jedem Start und nach geladenem Workspace kann Code ausgeführt werden. Der Einsprungpunkt ist die Methode `earlyStartup()` der Schnittstelle `org.eclipse.ui.startup.IStartup`.

Diese Schnittstelle wird in der Klasse `StartupToolConfigurator` implementiert. Die implementierte Methode überprüft ob alle zu überwachenden Konfigurationen noch existent sind, falls dem nicht so ist erstellt sie einen Wizarddialog aus einer Instanz der Klasse `StartupWizard`.

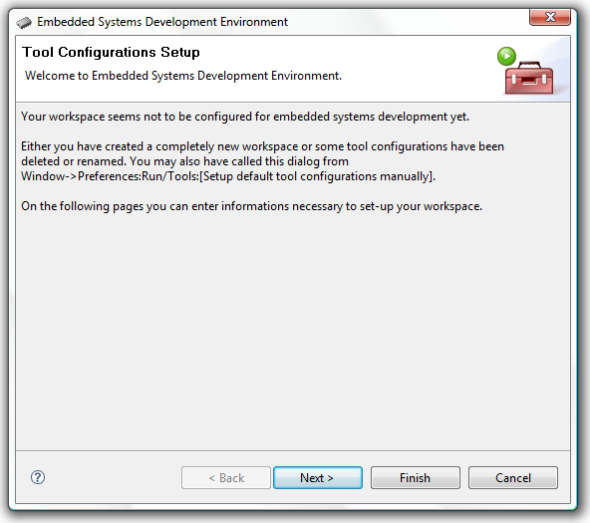


Bild 3.10: Seite 1 des Startup-Wizard

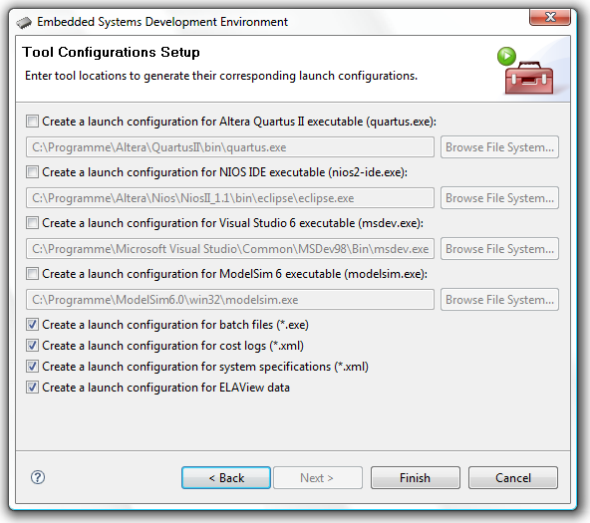


Bild 3.11: Seite 2 des Startup-Wizard

Die erste Wizardseite `StartupWelcomePage` wird mit statischem Begrüßungs- und Einleitungstext gefüllt (Abbildung 3.10). Auf der zweiten Wizardseite `StartupConfigurationsPage` werden – ähnlich wie bei den Steuerelementen der Präprozessorvariablen von Projektschablonen im vorherigen Abschnitt – konfigurationsabhängig Steuerelemente eingerichtet und initialisiert (Abbildung 3.11). Hier fasst eine Schnittstelle `IStartupOption` alle Ausprägungen von Steuerelementen zusammen, derzeit sind es die Optionsfelder `StartupCheckOption` (eine beschriftete Checkbox) und `StartupFileOption` (ein Editfeld mit einem Button *Browse for file...* und einem damit verbundenen Dialog zur Dateiauswahl). Da die zweite Wizardseite `StartupConfigurationsPage` über Änderungen an den auf ihr enthaltenen Steuerelementen informiert werden möchte, implementiert sie die Schnittstelle `IStartupOptionListener` und registriert sich bei den Steuerelementen (sog. Beobachtermuster, [GHJV96, p. 287 ff.]).

Die Klasse `StartupUtil` bietet alle Funktionen zum Laden und Speichern von Konfigurationseinstellungen und zum Generieren von Konfigurationen an. Die Klasse `IStartupPreferences` dient als Behältnis für globale Konstanten, sie enthält die IDs der zu persistierenden Optionen.

Das Klassendiagramm in Abbildung 3.12 visualisiert die wichtigsten Zusammenhänge.

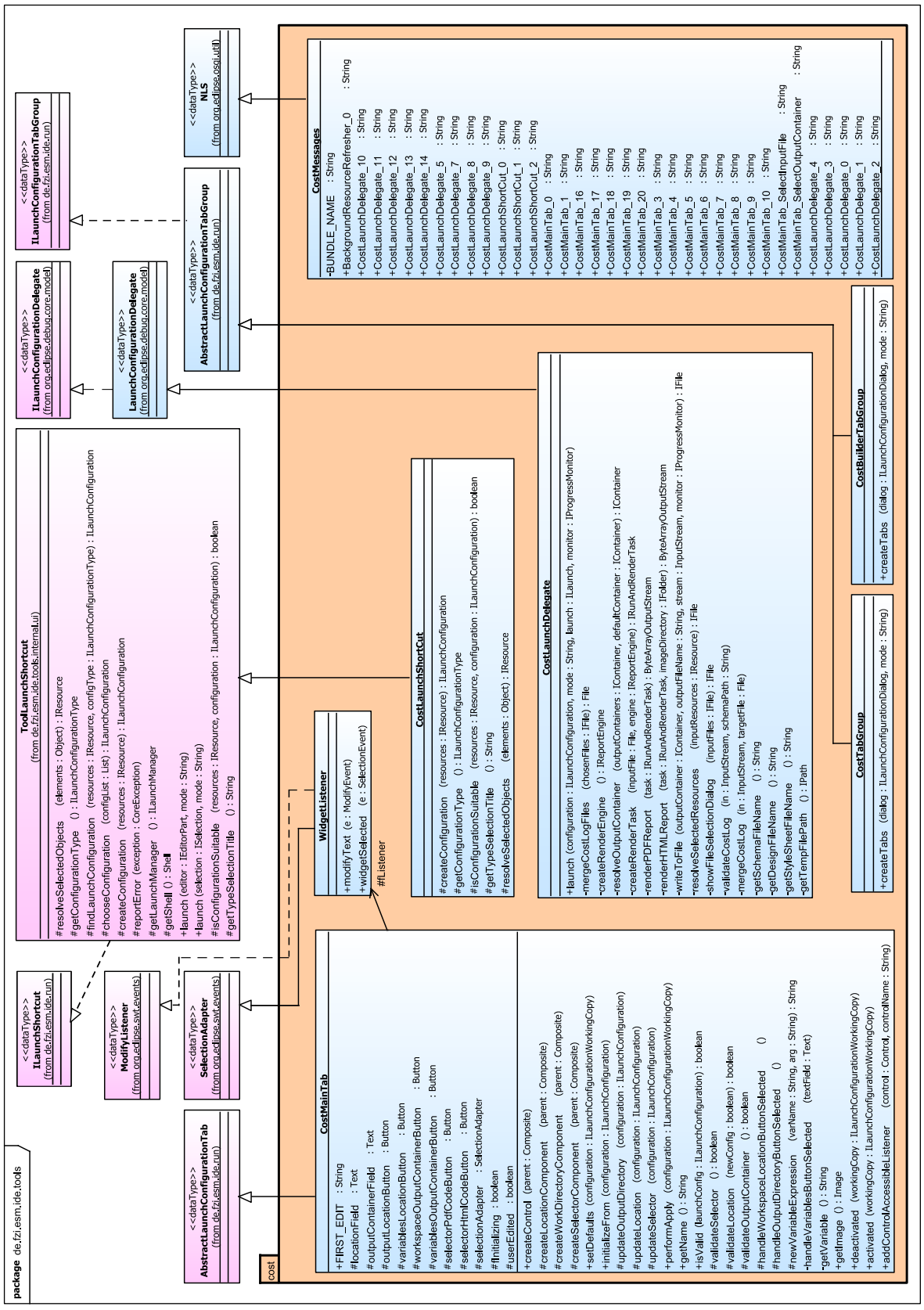


Bild 3.6: Vererbungshierarchie bei der Implementierung einer Werkzeugart

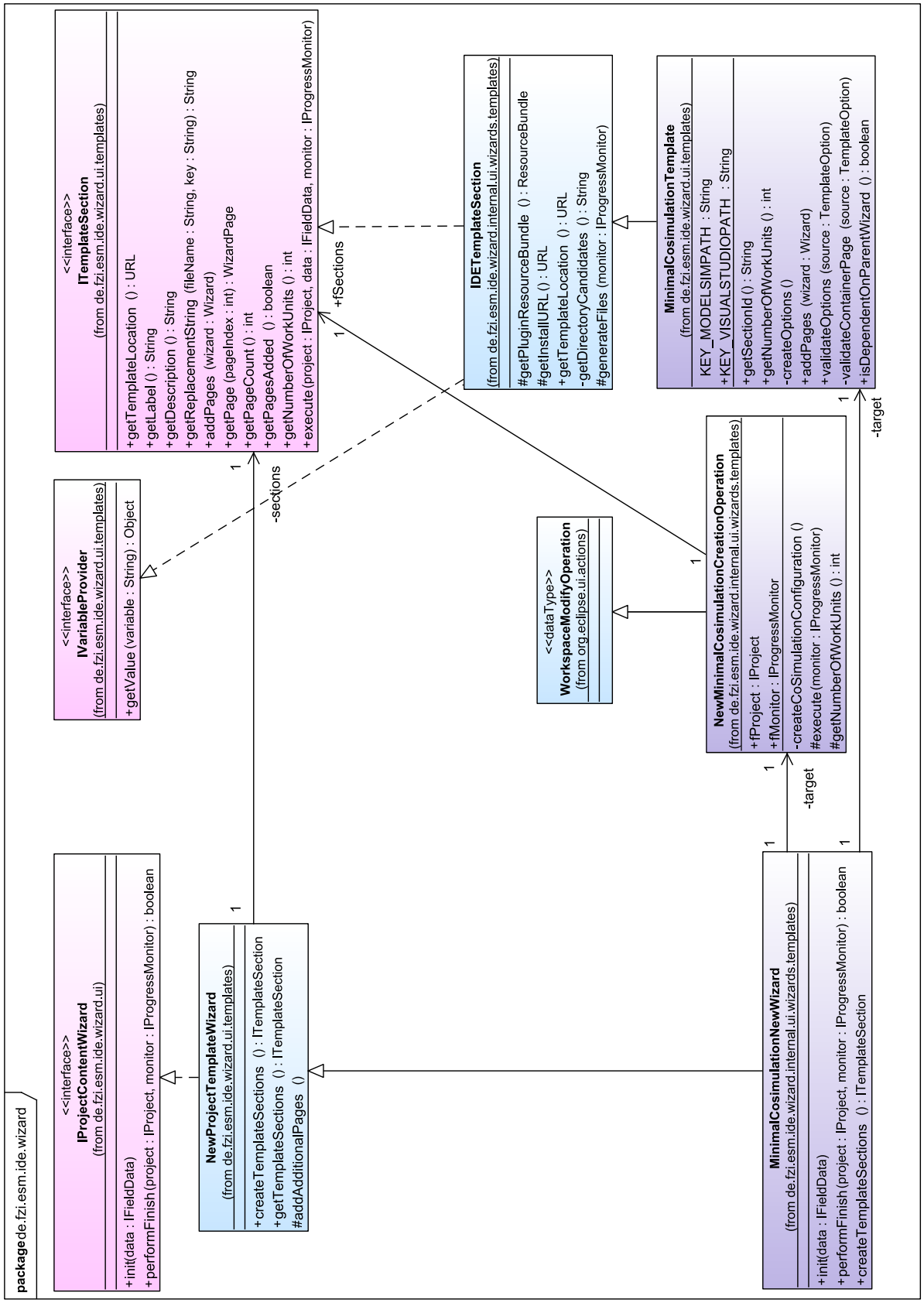


Bild 3.9: Vererbungshierarchie bei der Implementierung einer Projektvorlage

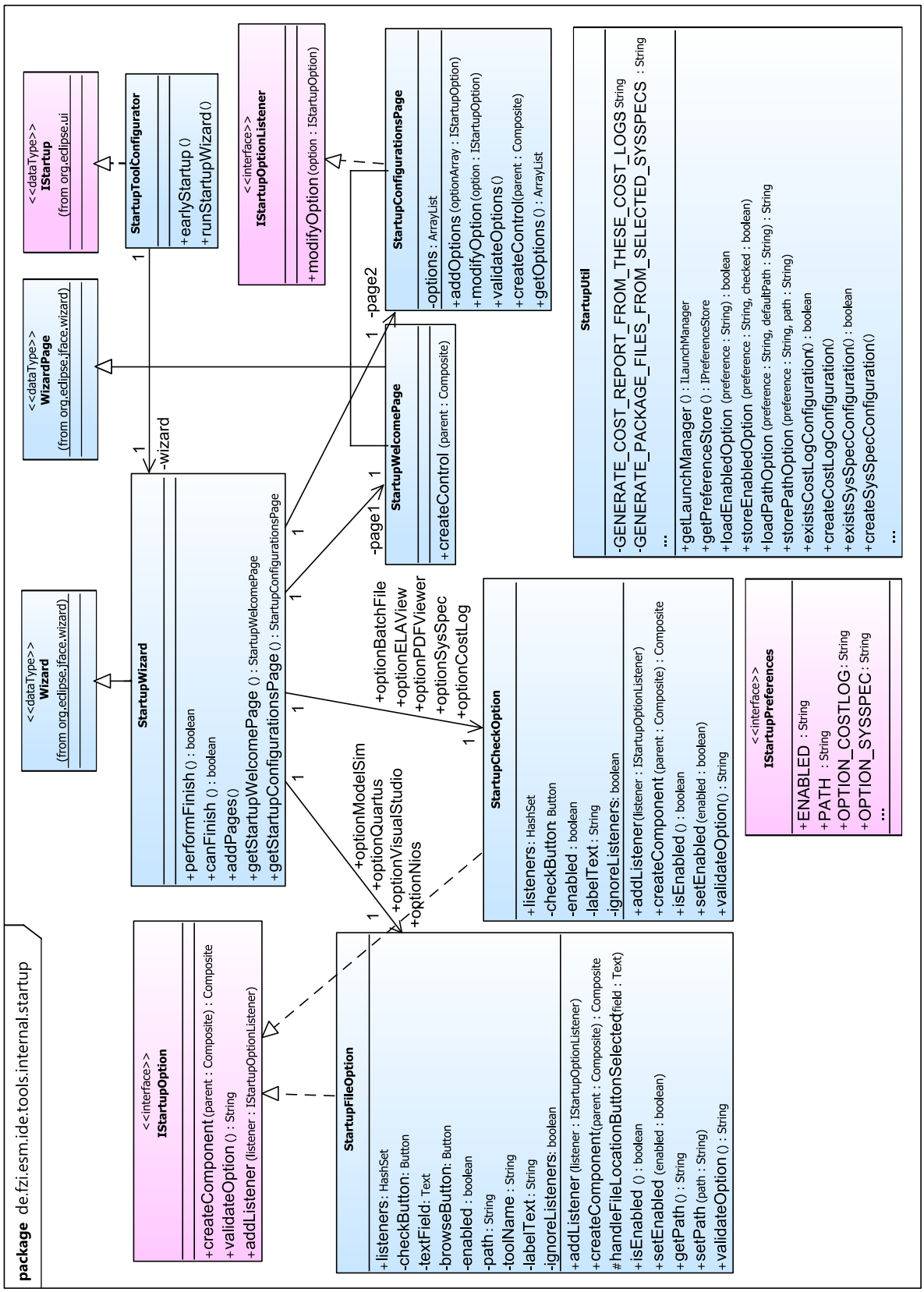


Bild 3.12: Wizard zur Pflege der Konfigurationen der Standardtools

4 Evaluierung

Die Arbeit endet mit einem Rückblick, einem Erfahrungsbericht über erste Erfahrungen, die bei der Arbeit mit dem System gemacht werden konnten, sowie einem Ausblick auf die Zukunft des Systems.

4.1 Rückblick

In Kapitel 1 wurden die Aufgabenstellung, das Umfeld, aus dem die Aufgabenstellung entspringt, und die geplanten Schritte zur Lösung skizziert.

Der erste Schritt, geschildert in Kapitel 2, bestand aus einer Analysephase. An Hand eines Szenarios wurde klar, welchen Anforderungen das zu entwerfende System gerecht werden muss, und dass sich derzeitige Lösungsansätze nur unzureichend diesen Anforderungen zu stellen vermögen. Der entscheidendste Punkt in der Konzeption war die Wahl einer Plattform, auf der ein geeigneteres System realisiert werden konnte. Die Entscheidung fiel auf die Eclipse Plattform, nicht zuletzt auf Grund ihrer hervorragenden Erweiterbarkeit.

Im zweiten Schritt fand die Realisierung des Systems statt. Kapitel 3 beschreibt die der Entwurfsphase entsprungenen Architektur im Groben. Interessante Teile der Architektur, insbesondere solche, die die Erweiterbarkeit des Systems garantieren, wurden genauer beleuchtet.

4.2 Bewertung erster Erfahrungen

Schon während der Entwicklung wurde das System in dem in Kapitel 2 beschriebenen Szenario eingesetzt. Die hierbei erlangten Erfahrungen und erkannten Fehler führten zu einer besseren Praktikabilität und Anwendbarkeit in späteren Iterationen.

Als Beispiel hierfür sei die Generierung von Kostenberichten genannt, die iterativ soweit verfeinert werden konnten, dass sie aus simulativen Ergebnissen gute Rückschlüsse auf die Qualität

der im Szenario beschriebenen Hardware-/Software-Architektur ziehen konnten. Anhang C illustriert dies an einem Beispiel.

Als weiteres Beispiel sei auch die Erstellung von Beispielprojekten genannt. Die schon jetzt implementierte Projektschablone *Minimal Cosimulation Project* illustriert, wie einfach ein Projekt mit Parametern und einer kleinen Menge Code für die dialogorientierte Einstellung dieser Parameter und Generierung der Projektartefakte ausgestattet werden kann, und dadurch zur Vorlage für neue Projekte dienen kann.

Der Einsatz des Metatools hat und wird sich im weiteren Einsatz als äußerst praktikabel erweisen. Gerade durch die Erweiterbarkeit können neue Werkzeuge hinzugefügt werden und vorhandene angepasst werden, das System ist dadurch flexibel und schnell an den Entwurfsprozess anpassbar und optimierbar.

Als größten Vorteil in der Anwendung der unter dem Dach des Metatools ESDE vereinten *Toolchain* ist die Zeitersparnis zu nennen, die sich durch die konsequente Anwendung des *Left-Shift*-Prinzips bei der Ausarbeitung des ESDE-Entwurfsprozesses ergeben hat. Dieses, von Andrew Farncombe formulierte Prinzip besagt, dass das Aufspüren und Eliminieren von Problemen möglichst früh (proaktiv) und nicht erst zum Ende des Entwicklungssprozesses (reaktiv) geschehen sollte, da die Kosten für das *Debugging* exponentiell mit der Zeit wachsen (Abbildung 4.1).

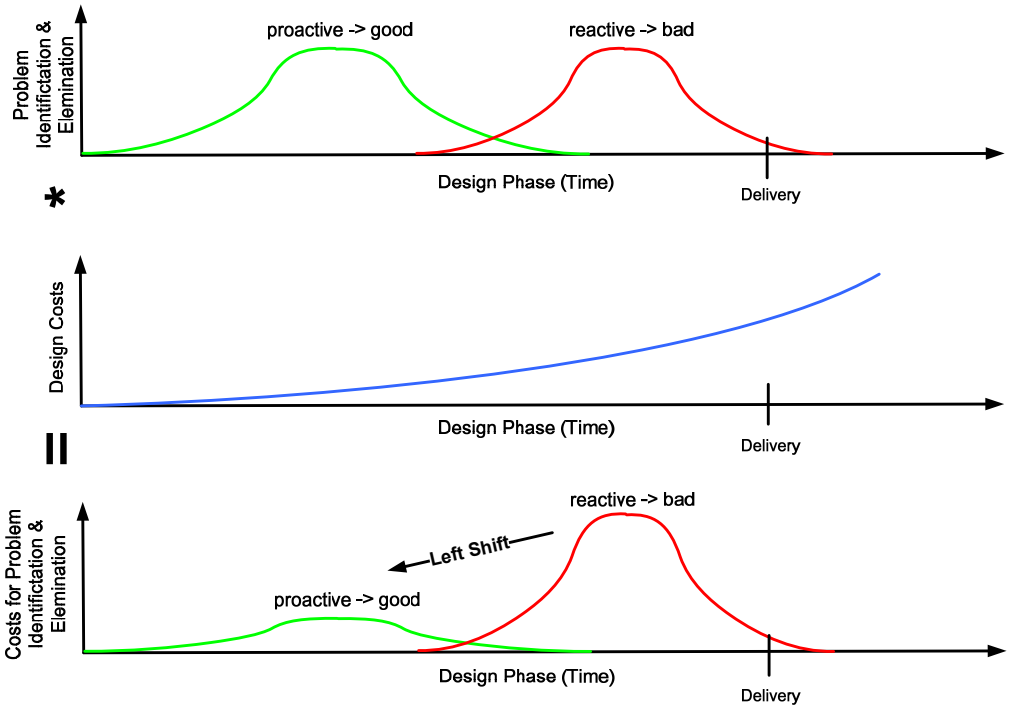


Bild 4.1: Das *Left-Shift*-Prinzip von Andrew Farncombe

4.3 Ausblick

Obgleich die wichtigsten Anforderungen von ESDE in seiner momentanen Entwicklungsstufe erfüllt werden, gibt es dennoch Punkte, an denen Verbesserungen und Erweiterungen angesetzt werden können. Zukünftige Entwicklungen können folgende Erweiterungen beinhalten:

- Eine **Dokumentation** (Handbuch) des Systems: Obwohl schon diese Arbeit einen gewissen Einblick in Funktion und Bedienung des Systems gewährt (nicht zuletzt beim Lesen des Pflichtenhefts in Anhang A), muss eine umfassende Dokumentation des Systems noch geschrieben werden. Hierbei soll – wie in der Industrie üblich – das Pflichtenheft als Vorlage dienen.
- Ein **Hilfesystem** mit integrierter Dokumentation: Rund um ESDE als Hardware-/Software-Entwurfsprozess und -Entwurfssystem existieren mehrere Quellen. Die Einbindung dieser Quellen in das Hilfesystem von Eclipse ist hauptsächlich redaktionelle Arbeit, da Hilfeplugins aus der HTML-Sonderform `.xhtml` bestehen und keinen Javacode erforderlich machen (mehr dazu in [DFK+05, pp. 547-562]). Bereits vorgesehen sind kleine Buttons mit einem Fragezeichen-Icon in den Dialogen in der linken unteren Ecke. Sie besitzen eine eigene ID und müssen nur noch im Hilfesystem verlinkt werden.
- Das Erstellen einer **Willkommensseite** mit Tutorials beim ersten Start: Eine Willkommensseite begrüßt den unerfahrenen Nutzer beim ersten Start und bietet ihm *Tutorials*, *Cheat Sheets* und einen Einstieg ins Hilfesystem. Eine solche Startseite kann hinzugefügt werden, indem das Plugin `org.eclipse.ui.intro` inkludiert und erweitert wird, *Cheat Sheets* werden durch Inklusion und Erweiterung von `org.eclipse.ui.cheatsheets` realisiert.
- Die Plugins des Systems als **Feature** verpacken: Die eigenständig entwickelten Plugins `de.fzi.esm.ide.*` bilden den Systemkern und ergeben am meisten Sinn, wenn sie zusammen installiert werden. In solchen Fällen sieht die Eclipseplattform eine Bündelung mehrerer Plugins zu einer Installationseinheit, einem *Feature* vor. Ein Feature kann mit einem eigenen Icon versehen werden, das innerhalb des About-Dialogs angezeigt wird, und erlaubt die bessere Identifikation des Produkts.
- Das System auf einer **Downloadseite** anbieten: Diese Änderung vereinfacht die Wartbarkeit, da das Deployment der neuesten Version automatisiert durchgeführt werden kann.
- Verbesserungen am **Layout** und an der **Auswertungslogik** des Kostenberichts: Eine Verbesserung der Auswertungslogik geschah während der Enrtwicklungszeit fortwährend, da ständig neue Erfahrungen gemacht wurden. Auch weiterhin kann der Kostenbericht

durch Editieren der Datei `costLog.rptdesign` verfeinert werden, und lässt damit zu, dass ohne Änderung des Javacodes neue Erfahrungen in das System einfließen können. Bereits während der Anfertigung dieser Studienarbeit begannen Praktikanten am FZI mit der Verbesserung des Layouts und der Verfeinerung des Kostenberichts.

A Pflichtenheft

Aus den Forderungen an das Produkt, die in Kapitel 2 aufgestellt wurden, wird ein *Pflichtenheft* abgeleitet. Das Pflichtenheft fasst alle Anforderungen an das zu entwerfende Softwareprodukt zusammen, es liefert die vertragliche Grundlage für alle späteren Entwurfs- sowie Implementierungsentscheidungen. Seine Gliederung lehnt sich an die von Helmut Balzert vorgeschlagene Gliederung [Bal98] an.

A.1 Zielbestimmung

Ein übergeordnetes Entwurfswerkzeug für eingebettete HW/SW-Systeme ist zu entwerfen und in den bestehenden Entwicklungsprozess einzugliedern. Das Produkt soll helfen, alle während des Entwurfs relevanten Artefakte in einer Projektstruktur zu ordnen, und alle während des Entwurfs zu verwendenden Werkzeuge für die direkte Anwendung zu konfigurieren. Augenmerk soll auf die intuitive Erweiterbarkeit um eigene Werkzeuge gerichtet werden.

A.1.1 Musskriterien

- Hierarchische Verwaltung von Daten, zusammengefasst zu Projekten.
- Aufruf externer Werkzeuge und Entwicklungsoberflächen auf Knopfdruck unter Einbeziehung des Aufrufkontextes (Aufruf im in der Projekthierarchie ausgewählten Verzeichnis, Übergabe der in der Projekthierarchie ausgewählten Dateien als Parameter).
- Vom Auftraggeber selbst entwickelte, kleinere Werkzeuge (z.B. Konverter, Generatoren) sind auf selektierte Daten anwendbar.
- Die Erstellung von Projekten erfolgt auf der Basis von Projektvorlagen. Inhalte gegebener Dateien können offen gelassen werden und bei der dialoggesteuerten Erstellung in Form von Optionen spezifiziert werden.
- Text und Code kann in einem Multipageeditor editiert werden.

- Bei der Oberfläche handelt es sich um eine gängige grafische Entwicklungsoberfläche, die spezialisierbar ist, eine Projektsicht hat und mehrere Dateien gleichzeitig per Registernavigation editieren kann. VHDL-, C- und XML-Code wird syntaxspezifisch eingefärbt (sog. *Syntax Highlighting*).

A.1.2 Wunschkriterien

- Eigene Projektvorlagen können nachträglich hinzugefügt werden.
- Die Verwendung von *Visual Studio* oder *Eclipse* ist zu bevorzugen.
- Beim erstmaligen Start können die Pfade zu externen Werkzeugen global angegeben werden, und später auf Wunsch in den Einstellungen geändert werden.
- Berechnete Messwerte aus der Simulation werden mit Hilfe eines *Business Intelligence Tools* grafisch aufbereitet und als Graphen angezeigt.

A.1.3 Abgrenzungskriterien

- Es ist kein vollständig neues Werkzeug zu implementieren. Vielmehr sollen bereits vorhandene, erweiterbare Entwicklungsplattformen an die Anforderungen angepasst werden. Dies garantiert die spätere Erweiterbarkeit und schränkt den Grad der Komplexität ein, die Menge zu wartenden Codes wird erheblich reduziert.

A.2 Produkteinsatz

Die Software wird im Entwicklungsprozess zur Entwicklung von eingebetteten Systemen eingesetzt.

A.2.1 Anwendungsbereiche

Das System dient der Erstellung eingebetteter HW/SW-Systeme.

A.2.2 Zielgruppen

Das System zielt auf Entwickler eingebetteter HW/SW-Systeme ab, die ihr Projekt übersichtlich verwalten wollen. Hauptzielgruppe sind im Speziellen Entwickler, die die vorgesehenen Werkzeuge verwenden (ModelSim, Quartus, Nios, Visual Studio 6).

A.2.3 Betriebsbedingungen

Die Software soll sich in den Betriebsbedingungen von anderen grafischen Entwicklungsumgebungen nicht unterscheiden.

physikalische Umgebung des Systems: gewöhnlicher Büroarbeitsplatzrechner

tägliche Betriebszeit: täglich, 24 Stunden

Wartung: möglichst keine

Beobachtung: unbeaufsichtigter Betrieb

A.3 Produktübersicht

Im System sind unterschiedliche Aktoren involviert (Abbildung A.1). Zum Einen ist dies der reguläre Anwender der Oberfläche, er verwendet die vorhandene Funktionalität zur Entwicklung eigener Systeme. Eine dedizierte Person (entweder der Anwender selbst oder eine für die Aufgabe speziell ausgewählte Person) kann das System um eigene Funktionen erweitern. Bei einigen Erweiterungen ist der Eingriff in den Quellcode notwendig. Ein vollständiges Neuerstellen (*Rebuild*) des Systems ist jedoch nicht erforderlich, da Erweiterungen als eigenständige Eclipse-Plugins erstellt werden können und durch einfaches Kopieren in den Eclipse-Plugin-Ordner eingepflegt werden.

A.4 Produktfunktionen

Alle verfügbaren Funktionen betreffen den Anwender, aus seiner Sicht werden folgende Funktionen bereitgestellt, die Übersicht dieser Funktionen ist im entsprechenden Geschäftsprozessdiagramm A.2 dargestellt:

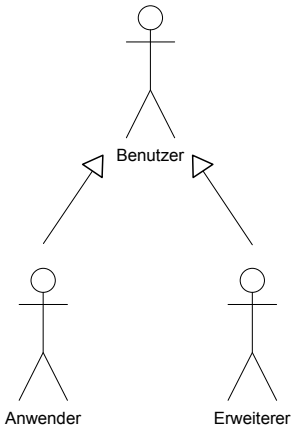


Bild A.1: Übersicht über die Aktoren im System

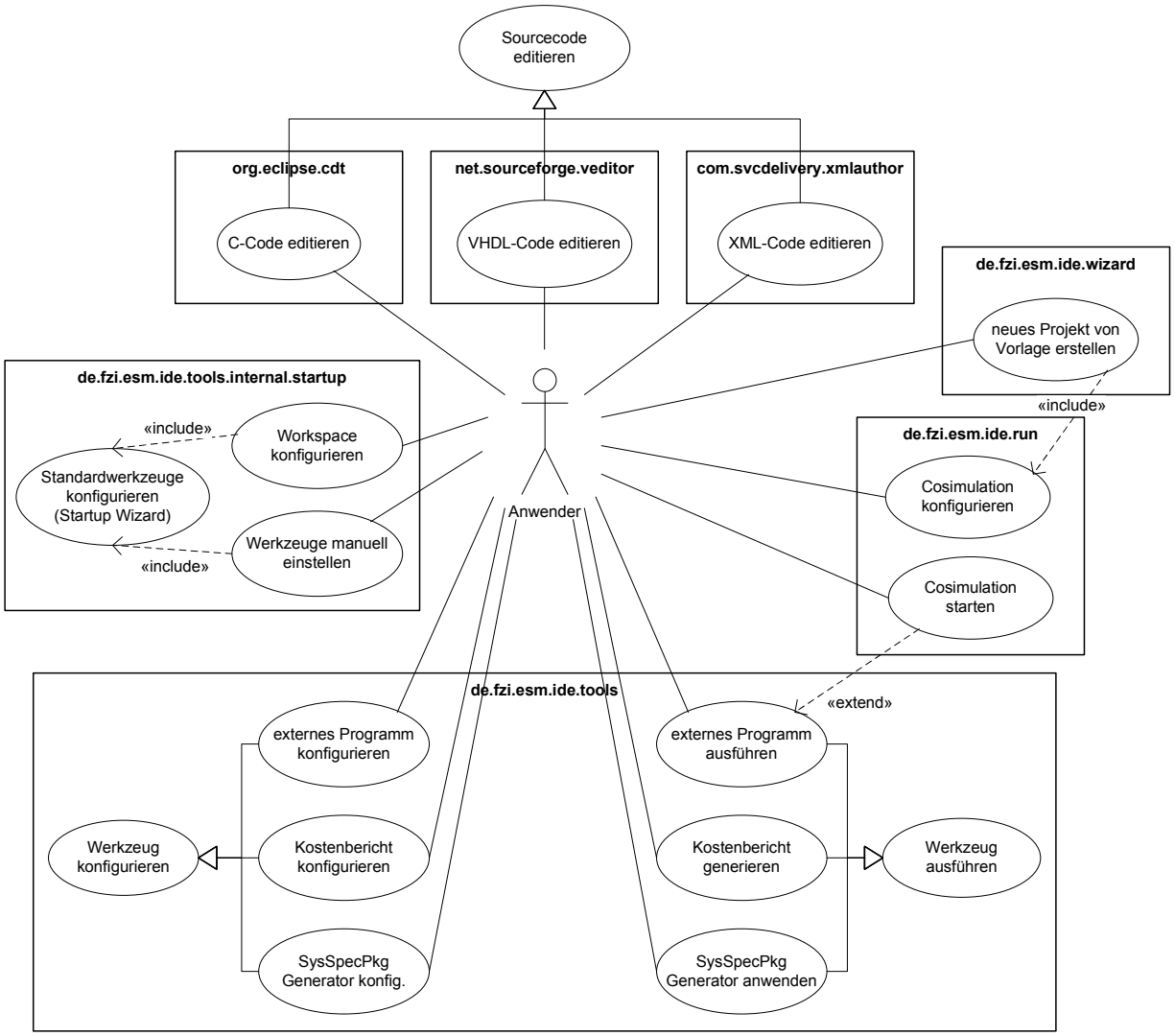


Bild A.2: Geschäftsprozesse des Anwenders

Eclipse stellt Standardfunktionen für Workbench- und Projektverwaltung zur Verfügung. Die Verfügbarkeit dieser Funktionalität wird vorausgesetzt, es wird lediglich auf Funktionen eingegangen, die nicht bereits von der integrierte Entwicklungsumgebung (IDE) implementiert werden.

/F0010/ Konfigurieren der Standardwerkzeuge beim ersten Start

Ziel Ziel ist die Einrichtung des Workspace beim ersten Start sowie die Erhaltung erfolgter Einstellungen bei weiteren Starts. Der Benutzer kann aus einer fest vorgegebenen Menge von Werkzeugen wählen, für die entsprechende Werkzeugkonfigurationen im Workspace eingerichtet werden sollen. Weiters soll bei einem bereits konfigurierten Workspace überprüft werden, ob die Werkzeugkonfigurationen zum jetzigen Zeitpunkt noch bestehen.

Vorbedingung Ein Workspace wurde beim Start ausgewählt, für den die Konfigurationen erstellt werden sollen. Die Auswahl kann auch implizit erfolgen, eine Nichtauswahl ist von Seiten der Basis-IDE nicht möglich, daher ist diese Bedingung immer erfüllt.

Nachbedingung Erfolg Der Nutzer hat all diejenigen Werkzeuge abgewählt, die er nicht zur Verfügung hat, und für die angewählten Werkzeuge eine existierende ausführbare Datei angeben können. Für diese Werkzeuge sind Konfigurationen eingerichtet worden.

Nachbedingung Fehlschlag Der Nutzer hat die Konfiguration entweder freiwillig oder gezwungenermaßen abgebrochen, weil er den Dialog auf Grund von nicht angegebenen oder nicht existierenden ausführbaren Dateien nicht bestätigen kann. Die gesamte IDE wird beendet.

Auslösendes Ereignis Bei jedem Start der Entwicklungsumgebung.

Voreinstellung Alle möglichen Werkzeuge sind zur Überwachung selektiert.

Beschreibung Der Startup-Wizard ermöglicht dem Nutzer das bequeme Einrichten von Konfigurationen, die sonst für mit dem System nicht vertraute Nutzer nur mühsam von Hand einzurichten sind. Die Überprüfung bei jedem Start hilft dem Nutzer zusätzlich, falls er versehentlich eine der Konfigurationen gelöscht hat. Folgende externen Werkzeuge werden konfiguriert:

- Altera Quartus II,
- NIOS II IDE,

- Visual Studio 6,
- ModelSim 6,
- Tool zur Ausführung von Stapeldateien,
- Datenvisualisierungstool ELAView.

Folgende im System selbst implementierten Werkzeuge werden vorkonfiguriert:

- Tool zur Generierung von Kostenberichten,
- Tool zur Generierung von Systemspezifikationen.

/F0020/ Manuelles, nachträgliches Konfigurieren der Standardwerkzeuge

Ziel Ziel ist das manuelle Ändern der Einstellungen des Workspaces. Der Benutzer kann aus einer fest vorgegebenen Menge von Werkzeugen wählen, für die entsprechende Werkzeugkonfigurationen im Workspace eingerichtet werden sollen. Weiters soll bei einem bereits konfigurierten Workspace überprüft werden, ob die Werkzeugkonfigurationen noch bestehen.

Vorbedingung Ein Workspace wurde beim Start ausgewählt, für den die Konfigurationen erstellt werden sollen. Die Auswahl kann auch implizit erfolgen, eine Nichtauswahl ist von Seiten der Basis-IDE nicht möglich, daher ist diese Bedingung immer erfüllt.

Nachbedingung Erfolg Der Nutzer hat all diejenigen Werkzeuge abgewählt, die er nicht zur Verfügung hat, und für die angewählten Werkzeuge eine existierende ausführbare Datei angeben können. Für diese Werkzeuge sind Konfigurationen eingerichtet worden.

Nachbedingung Fehlschlag Der Nutzer hat die Konfiguration entweder freiwillig oder gezwungenermaßen abgebrochen, weil er den Dialog auf Grund von nicht angegebenen oder nicht existierenden ausführbaren Dateien nicht bestätigen kann. Der Dialog wird geschlossen.

Auslösendes Ereignis Klick auf einen Button unter *Window – Preferences* im Abschnitt *Run – Tools*.

Voreinstellung Alle Werkzeuge sollen konfiguriert werden.

Beschreibung Der manuelle Aufruf des Startup-Wizards von /F0010/ erlaubt dem Nutzer das nachträgliche Ändern und Umkonfigurieren seiner Werkzeugkonfigurationen.

/F0030/ Erstellung eines Projektes von einer Vorlage

Ziel Der Prozess zielt auf die Einrichtung eines Projektes von einer vorgegebenen Auswahl von Schablonen ab. Aus einem Dialog lässt sich eine Schablone wählen und einem weiteren Schritt von ihr benötigte Parameter einstellen.

Vorbedingung Der Workspace besitzt noch kein gleichnamiges Projekt, der Name des Projektes ist nach Eclipseregeln gültig. Alle Parameter wurden korrekt eingegeben.

Nachbedingung Erfolg Im Workspace wird ein Projekt samt einer Konfiguration zum Start der Kosimulation eingerichtet.

Nachbedingung Fehlschlag Bei nicht akzeptiertem bzw. bereits vorhandenem Projektnamen lässt sich der Dialog nicht abschließen. Falsch eingegebene Parameter werden vom Dialog nicht akzeptiert oder führen zu einem fehlerhaft erstellten Projekt und zu fehlerhaft erstellten Konfigurationen.

Auslösendes Ereignis Der Nutzer wählt *File – New – Project...* und im erscheinenden Dialog *Embedded Development – Embedded System Project from Template* aus.

Voreinstellung –

Beschreibung Der Prozess übernimmt das automatisierbare Einstellen von Projektkonfigurationen. Eine Schablone für ein Minimalprojekt hilft Nutzern Zeit zu sparen. Dem unerfahrenen Nutzer helfen erweiterte Schablonen beim Einstieg in die Oberfläche, indem sie verfügbare Funktionen demonstrieren. Für Erweiterer besteht die Möglichkeit, eigene Projektschablonen dem Prozess hinzuzufügen.

/F0040/ Editieren von C-, VHDL- oder XML-Code

Ziel Ziel des Prozesses ist das Ändern von Projektdateien die Quellcode vom entsprechenden Typ beinhalten.

Vorbedingung Im Workspace befinden sich Projektdateien mit entsprechender Dateiendung (`.c`, `.h`, `.vhd`, `.xml`)

Nachbedingung Erfolg Die Datei wird in den Editor geladen. Entsprechende Features der Eclipseplattform unterstützen den Nutzer beim Edieren des Inhalts. Darunter fällt die farbliche Hervorhebung der syntaktischen Elemente, ein Fenster mit einer Übersicht der syntaktischen Struktur, sowie eine automatische Codevervollständigung bei Schlüsselwörtern und einfachen Strukturen (z.B. `vol` induziert `volatile`, `<p>` induziert `</p>`). Anschließend wird die Datei geschlossen und Änderungen dabei gespeichert.

Nachbedingung Fehlschlag Die Datei lässt sich Aufgrund von Zugriffsproblemen von Seiten des Betriebssystems nicht öffnen oder beim Schließen speichern.

Auslösendes Ereignis Das Öffnen einer Datei mit einer der genannten Dateiendungen.

Voreinstellung –

Beschreibung Wenngleich das Ändern von C- und VHDL-Code nicht Hauptaufgabe des Systems ist (dies übernehmen externe Werkzeuge wie Visual Studio und ModelSim), so soll es dennoch den Nutzer bei der Verwaltung von Sourcecode mittels Einsehen unterstützen. Da das System den Quelltext nicht kompilieren muss, entfällt ebenfalls die Anzeige von Fehlern. Anders bei XML-Dateien; hier muss das System möglichst schon während der Eingabe strukturelle Fehler anzeigen, da später Tools aus den Daten Artefakte generieren sollen.

/F0050/ Einrichten einer Kosimulationskonfiguration

Ziel Projektspezifische Angaben zur Ausführung einer Simulation können gemacht werden.

Vorbedingung Das entsprechende Projekt befindet sich im aktuellen Workspace. Die zu verwendenden Tools zur Simulation sind lokal installiert.

Nachbedingung Erfolg Eine ausführbare Konfigurationseinheit ist eingerichtet, die das gewünschte Projekt ausführen kann.

Nachbedingung Fehlschlag Bei unverträglichen Parametern kann der Dialog nicht bestätigt werden oder führt zu einem Fehler bei der Ausführung.

Auslösendes Ereignis Der Dialog wurde z.B. über *Run – Run…* geöffnet, *Cosimulation* wurde ausgewählt. Anschließend wurde mit *New Launch Configuration* eine neue Konfiguration erstellt oder eine existierende Konfiguration ausgewählt. Nach dem Ändern der Einstellungen wurde das Speichern durch einen Druck auf *Apply* ausgelöst.

Voreinstellung Beim Einrichten über die minimale Standardprojektschablone existiert eine voreingestellte Konfiguration bereits.

Beschreibung Es müssen der Name des Programms das die Hardwaredefinition simuliert und der Name des Programms das die auf dieser Hardware ablaufende Software in einem Debugger simuliert angegeben werden. Ausführungsverzeichnis und Argumente können für das jeweilige Programm zusätzlich spezifiert werden. Dabei können eine Reihe von dynamischen Variablen helfen. Die Verwendung orientiert sich

weitestgehend an den von der Java Development Tools (JDT) eingeführten Variablen:

- `${workspace_loc:/Workspacepfad/zu/Ressource}` löst einen Eclipse-internen Workspace-relativen Pfad einer Ressource in einen absoluten Pfad auf, der der entsprechenden Ressource im globalen Dateisystem entspricht. So könnte

  ```
  ${workspace_loc:/TestProject1/Simulation/}
  ```

  etwa auf

  ```
  C:\Eclipse\My Workspace\TestProject1\Simulation\
  ```

  abgebildet werden. Wird lediglich `${workspace_loc}` angegeben, wird der absolute Pfad zum Wurzelverzeichnis des Workspace zurückgegeben.

- Die Variable `${resource_loc:/Workspacepfad/zu/Ressource}` steht für den entsprechenden Workspace-relativen Pfad zur angegebenen Datei bzw. Verzeichnis. Die Variable `${resource_loc}` allein repräsentiert diejenigen Pfade zu Dateien/Verzeichnissen relativ zum Workspace, die zur Anwendungszeit der Konfiguration im Workspace selektiert sind.

- Die Variable `${resource_path:/Workspacepfad/zu/Ressource}` berechnet den absoluten Pfad im globalen Dateisystem zu der relativ zum Workspace angegebenen Datei bzw. Verzeichnis. Bei fehlendem Argument berechnet sie diejenigen Pfade zu Dateien/Verzeichnissen im globalen Dateisystem, die zur Anwendungszeit der Konfiguration im Workspace selektiert sind.

- Die Variable `${resource_name:/Workspacepfad/zu/Ressource}` repräsentiert den Namen zu der relativ zum Workspace angegebenen Datei bzw. Verzeichnis. Ohne Argument repräsentiert sie diejenigen Namen von Ressourcen, die zur Anwendungszeit der Konfiguration im Workspace selektiert sind.

- Die Variable `${container_loc:/Workspacepfad/zu/Ressource}` steht für den entsprechenden Workspace-relativen Pfad zum übergeordneten Verzeichnis der angegebenen Ressource. Argumentfrei repräsentiert sie diejenigen Pfade zu Verzeichnissen relativ zum Workspace, die zur Anwendungszeit der Konfiguration im Workspace selektierte Ressourcen direkt übergeordnet sind.

- Die Variable `${container_path:/Workspacepfad/zu/Ressource}` berechnet den absoluten Pfad im globalen Dateisystem zu demjenigen Verzeichnis, das die relativ zum Workspace angegebene Ressource (Datei oder Verzeichnis) enthält.

Die Variable `${container_path}` alleine berechnet diejenigen absoluten Pfade zu Verzeichnissen im globalen Dateisystem, in denen sich die im Workspace selektierten Ressourcen befinden.

- Die Variable `${container_name:/Workspacepfad/zu/Ressource}` repräsentiert den Namen zum Verzeichnis, das die relativ zum Workspace angegebene Ressource enthält. Ohne Argument repräsentiert sie diejenigen Namen von Verzeichnissen, die zur Anwendungszeit der Konfiguration im Workspace selektierten Ressourcen direkt übergeordnet sind.

Die Angabe von auflösenden Variablen kann dabei helfen, für ein spezielles Projekt Konfigurationen zu erstellen, indem Workspace-interne Pfade in Pfade des globalen Dateisystem des Betriebssystems aufgelöst werden können. Weiterhin ist es möglich, Variablen einzusetzen. Optional ist beim Beenden einer Simulation das Auffrischen der Workspace-Ressourcen möglich, um Änderungen sichtbar machen zu können. Umgebungsvariablen können angegeben werden und die Anzeige der Befehlskonsole kann aktiviert werden.

/F0060/ Starten eines Projekts über eine Kosimulationskonfiguration

Ziel Für ein Projekt wird eine Kosimulationssitzung begonnen.

Vorbedingung Ein Projekt und eine entsprechende Konfiguration für das Projekt (siehe /F0050/) existieren.

Nachbedingung Erfolg Die beiden Anwendungen zur Simulation von Hardware und Software wurden gestartet und haben die übergebenen Argumente akzeptiert. Nach Beendigung beider Prozesse wurden – falls eingestellt – die geänderten Ressourcen im Arbeitsplatz aktualisiert.

Nachbedingung Fehlschlag Ein Dialog wird geöffnet mit der Benachrichtigung, dass Probleme bei der Interpretation von Variablen aufgetreten sind. Bei aktivierter Konsolenansicht können Fehler bei der Ausführung der Tools eingesehen werden.

Auslösendes Ereignis Es wurde z.B. der Konfigurationsdialog für Ausführungen über *Run – Run...* geöffnet, *Cosimulation* wurde ausgewählt. Anschließend wurde eine existierende Konfiguration ausgewählt und die Ausführung durch *Run* eingeleitet. Alternativ ist die kontextsensitive Ausführung über Rechtsklick auf eine Datei im Arbeitsplatz und die anschließende Auswahl einer Konfiguration im Untermenü *Run* möglich.

Voreinstellung Beim Einrichten über die minimale Standardprojektschablone existiert eine voreingestellte Konfiguration bereits. Sie startet ModelSim auf der Hardware- und Visual Studio 6 auf der Software-Seite.

Beschreibung Es werden alle vorhandenen Variablen interpretiert. Die angegeben Programme werden auf der Konsole des Betriebssystems ausgeführt. Parameter werden dazu übergeben, Umgebungsvariablen gesetzt, die Ausgabe während der Ausführung evtl. auf das jeweilige Konsolenfenster in Eclipse umgeleitet. Nach dem Beenden beider Prozesse werden geänderte Ressourcen falls gewünscht im Projektbaum aktualisiert.

/F0070/ Konfigurierung eines externen Programms

Ziel Workspace-spezifische Angaben zur Ausführung eines externen Werkzeugs können gemacht werden.

Vorbedingung Das zu verwendende externe Tool ist lokal installiert.

Nachbedingung Erfolg Eine ausführbare Konfigurationseinheit ist eingerichtet, die das externe Tool ausführen kann.

Nachbedingung Fehlschlag Bei unverträglichen Parametern kann der Dialog nicht bestätigt werden oder führt zu einem Fehler bei der Ausführung.

Auslösendes Ereignis Der Dialog wurde z.B. über *Run – Tools... – Tools...* geöffnet, *Program* wurde ausgewählt. Anschließend wurde mit *New Launch Configuration* eine neue Konfiguration erstellt oder eine existierende Konfiguration ausgewählt. Nach dem Ändern der Einstellungen wurde das Speichern durch einen Druck auf *Apply* ausgelöst.

Voreinstellung Beim Einrichten des Workspace über den Start-Up-Wizard existieren voreingestellte Konfigurationen bereits.

Beschreibung Es muss der Name des auszuführenden Programms angegeben werden. Ausführungsverzeichnis und Argumente können zusätzlich spezifiert werden. Die Angabe von kontextabhängigen Variablen verhält sich genau so, wie in /F0050/ beschrieben. Optional ist beim Beenden einer Simulation das Auffrischen der Workspace-Ressourcen möglich, um Änderungen sichtbar machen zu können. Umgebungsvariablen können angegeben werden und die Anzeige der Befehlskonsole kann aktiviert werden.

/F0080/ Start eines externen Programms

Ziel Die Anwendung eines Werkzeugs auf eine Ressource im Arbeitsplatz.

Vorbedingung Das zu verwendende externe Tool ist lokal installiert, eine entsprechende Konfiguration wurde eingerichtet (/F0070/).

Nachbedingung Erfolg Das Tool wurde in einem eigenen Prozess gestartet. Nach Beendigung wurden – falls eingestellt – die geänderten Ressourcen im Arbeitsplatz aktualisiert.

Nachbedingung Fehlschlag Ein Dialog wird geöffnet mit der Benachrichtigung, dass Probleme bei der Interpretation von Variablen aufgetreten sind. Bei aktivierter Konsolenansicht können Fehler bei der Ausführung des Programms eingesehen werden.

Auslösendes Ereignis Der Konfigurations-Dialog für Werkzeuge wurde z.B. über *Run – Tools... – Tools...* geöffnet, *Program* wurde ausgewählt. Anschließend wurde eine existierende Konfiguration ausgewählt und die Ausführung durch *Run Tool* eingeleitet. Alternativ ist die kontextsensitive Ausführung über Rechtsklick auf eine Datei im Arbeitsplatz und die anschließende Auswahl einer Konfiguration im Untermenü *Apply Tool...* möglich.

Voreinstellung Beim Einrichten des Workspace über den Start-Up-Wizard existieren voreingestellte Konfigurationen bereits.

Beschreibung Alle vorhandenen Variablen werden interpretiert. Dann führt er die angegeben Programme auf der Konsole des Betriebssystems aus. Parameter werden dazu übergeben, Umgebungsvariablen gesetzt, die Ausgabe während der Ausführung evtl. auf das Konsolenfenster der Anwendung umgeleitet. Nach dem Beenden des Prozesses werden geänderte Ressourcen je nach Option im Projektbaum aktualisiert.

/F0090/ Konfigurierung des Generators für Berichte

Ziel Das Erstellen einer Konfiguration, mit der von ModelSim gemessene Simulationsdaten graphisch ausgewertet werden können.

Vorbedingung –

Nachbedingung Erfolg Eine Konfigurationseinheit ist im Workspace eingerichtet, mit der Berichte generiert werden können.

Nachbedingung Fehlschlag Bei unverträglichen Parametern kann die Einrichtung nicht abgeschlossen werden.

Auslösendes Ereignis Der Dialog wurde z.B. über *Run – Tools... – Tools...* geöffnet, *Cost Log Report* wurde ausgewählt. Anschließend wurde mit *New Launch Configuration* eine neue Konfiguration erstellt oder eine existierende Konfiguration ausgewählt. Nach dem Ändern der Einstellungen wurde das Speichern durch einen Druck auf *Apply* ausgelöst.

Voreinstellung Beim Einrichten des Workspace über den Start-Up-Wizard existiert bereits eine voreingestellte Konfiguration. Diese Konfiguration erstellt einen Bericht mit HTML- und PDF-Ausgabe. Eingabedateien sind die zum Ausführungszeitpunkt selektierten Dateien. Ausgabeverzeichnis des Berichts ist eben dieses Verzeichnis.

Beschreibung Eine XML-Eingabedatei bzw. ein Verzeichnis mit den XML-Eingabedateien und das Ausgabeverzeichnis können festgelegt werden, dabei besteht wie gewohnt die Möglichkeit, Variablen wie in /F0050/ beschrieben zu verwenden.

/F0100/ Erstellung eines Berichts an Hand von Simulationsdaten

Ziel Das Generieren eines graphischen Berichts aus von ModelSim gemessenen Simulationsdaten.

Vorbedingung Eine entsprechende Konfiguration (/F0090/) ist eingerichtet, Simulationsdaten für die Eingabe befinden sich innerhalb des geladenen Workspace.

Nachbedingung Erfolg Je nach Option ist ein Bericht im PDF-Format oder/und im HTML-Format ausgegeben worden.

Nachbedingung Fehlschlag Ein Dialog wird geöffnet mit der Benachrichtigung, dass Probleme bei der Interpretation von Variablen aufgetreten sind, oder dass keine der angegebenen Dateien schemakonform ist. Ebenfalls wird ein Fehler im Dialog präsentiert, wenn die Ausführung aufgrund BIRT-interner Probleme nicht zu Ende geführt werden konnte (z.B. nicht ausreichender Heap-Speicher).

Auslösendes Ereignis Der Konfigurations-Dialog für Werkzeuge wurde z.B. über *Run – Tools... – Tools...* geöffnet, *Cost Report Generator* wurde ausgewählt. Anschließend wurde eine existierende Konfiguration ausgewählt und die Ausführung durch *Run Tool* eingeleitet. Alternativ ist die kontextsensitive Ausführung über Rechtsklick auf eine Datei im Arbeitsplatz und die anschließende Auswahl einer Konfiguration im Untermenü *Apply Tool...* möglich. In beiden Fällen erscheint ein Dialogfenster mit allen angegebenen Dateien, die dem Schema entsprechen; die Auswahl kann eingeschränkt werden und mit *OK* bestätigt werden. Von vornherein sind alle Dateien ausgewählt.

Voreinstellung Beim Einrichten über die minimale Standardprojektschablone existiert eine voreingestellte Konfiguration bereits. Sie erstellt für die im Workspace ausgewählten Dateien einen Bericht in den Formaten PDF und HTML. Markierte Verzeichnisse werden rekursiv durchlaufen.

Beschreibung Eine Auswertung der Messdaten ist durchführbar. Sie zeigt, nach Simulationssitzungen getrennt, die statischen Daten tabellarisch und die dynamischen Daten in Graphen aufgeführt. Ein abschließender Quervergleich zeigt die von allen Sitzungen gemessenen dynamischen Daten, nach gemeinsamen Modulen getrennt, in jeweils einem Graphen.

/F0110/ Konfigurierung des Generators für Systemspezifikationen

Ziel Das Einrichten einer Konfigration, mit der Systemspezifikationen auf Hardware- und Softwareseite publiziert werden können.

Vorbedingung –

Nachbedingung Erfolg Eine Konfigurationseinheit ist im Workspace eingerichtet, mit der Berichte generiert werden können.

Nachbedingung Fehlschlag Bei unverträglichen Parametern kann der Vorgang des Einrichtens nicht abgeschlossen werden.

Auslösendes Ereignis Der Dialog wurde z.B. über *Run* – *Tools...* – *Tools...* geöffnet, *System Specification Package Generator* wurde ausgewählt. Anschließend wurde mit *New Launch Configuration* eine neue Konfiguration erstellt oder eine existierende Konfiguration ausgewählt. Nach dem Ändern der Einstellungen wurde das Speichern durch einen Druck auf *Apply* ausgelöst.

Voreinstellung Beim Einrichten des Workspace über den Start-Up-Wizard existiert bereits eine voreingestellte Konfiguration. Diese Konfiguration überlässt die Auswahl der entsprechenden Profile dem Anwender. Die erstellten Artefakte werden im dafür vorgesehenen Ordner des aktiven Projekt angelegt:

```
\TargetSources\SystemSpecification
```

Beschreibung Eine XML-Eingabedatei und das Ausgabeverzeichnis können festgelegt werden, dabei besteht wie gewohnt die Möglichkeit, Variablen wie in /F0050/ beschrieben zu verwenden. Welche Profile für die Generierung der Ausgabe in ANSI-Cünd VHDL verwendet werden, kann getrennt spezifiziert werden. Mehrere Profile

werden durch Komma separiert. Wird eins der Felder frei gelassen, muss der Anwender zur Anwendungszeit unter den verfügbaren Profilen wählen.

/F0120/ Erstellung einer Systemspezifikation

Ziel Systemspezifikationen auf Hardware- und Softwareseite verfügbar machen.

Vorbedingung Eine entsprechende Konfiguration (/F0110/) ist eingerichtet, Systemspezifikationen für die Eingabe befinden sich innerhalb des geladenen Workspace.

Nachbedingung Erfolg Aus der Spezifikation und den ausgewählten Profilen wurde ANSI-C- und VHDL-Code erstellt.

Nachbedingung Fehlschlag Ein Dialog wird geöffnet mit der Benachrichtigung, dass Probleme bei der Interpretation von Variablen aufgetreten sind, oder dass keine der statuierten Dateien schemakonform ist. Ebenfalls wird ein Fehler im Dialog präsentiert, wenn die Ausführung aufgrund interner Probleme des Parsers/Generators nicht zu Ende geführt werden konnte (z.B. nicht ausreichender Heap-Speicher).

Auslösendes Ereignis Der Konfigurations-Dialog für Werkzeuge wurde z.B. über *Run – Tools... – Tools...* geöffnet, *System Specification Package Generator* wurde ausgewählt. Anschließend wurde eine existierende Konfiguration ausgewählt und die Ausführung durch *Run Tool* eingeleitet. Alternativ ist die kontextsensitive Ausführung über Rechtsklick auf eine Datei im Arbeitsplatz und die anschließende Auswahl einer Konfiguration im Untermenü *Apply Tool...* möglich. Für ANSI-C- und VHDL-Ausgabe können jeweils in einem Dialog die zu verwendenden Profile ausgesucht werden.

Voreinstellung Beim Einrichten über die minimale Standardprojektschablone existiert eine voreingestellte Konfiguration bereits, mehr dazu siehe /F0110/.

Beschreibung Diese Funktion ermöglicht das Publizieren von globalen Systemkonstanten sowohl auf Hardware- als auch Softwareseite. C-Headerdatei und VHDL-Datei werden aus einer einem speziellen Schema folgenden XML-Daten heraus generiert.

A.5 Produktdaten

Eclipse stellt bereits eine Standardfunktionalität zur Verfügung, diese betrifft Einstellungen für die IDE sowie das Persistieren von Projekten und deren Einstellungen. Das Speichern dieser

Daten wirdvorausgesetzt, im Folgenden werden lediglich solche Daten besprochen, die durch die erweiterten Funktionen anfallen.

/D0010/ Auswahl der zu konfigurierbaren Standardwerkzeuge aus /F0010/. Diese werden somit bei jedem Start der Anwendung sowie beim manuellen Start des Startupwizards /F0020/ verifiziert.

/D0030/ Geänderte Editoreinstellungen für C-, VHDL- und XML-Editoren /F0040/. Z.B. Einfärbung der syntaktischen Elemente,

/D0040/ Erstellte Kosimulationskonfigurationen für die Funktionen /F0050/ und /F0060/.

/D0050/ Erstellte Konfigurationen der Standard-Werkzeuge für die Funktionen /F0070/ und /F0080/.

/D0060/ Erstellte Konfigurationen von externen Werkzeugen für die Funktionen /F0090/ und /F0100/.

/D0070/ Erstellte Konfigurationen des Generators für Berichte für die Funktionen /F0110/ und /F0120/.

/D0080/ Erstellte Konfigurationen des Generators für Systemspezifikationen für die Funktionen /F0130/ und /F0140/.

A.6 Produktleistungen

/L0010/ Generieren eines Berichts (/F0120/) aus 10000 Messdaten dauert unter 1 Minute.

/L0020/ Generieren von Systemspezifikationen (/F0140/) aus einem Skript kleiner 100 kB dauert unter 30 Sekunden.

A.7 Benutzungsoberfläche

In diesem Abschnitt werden die von den Funktionen vorgegebenen Dialogschnittstellen vorgestellt. Art und Aufbau der Dialoge sind dem Stil von Eclipse und der Java Development Tools (JDT)-Erweiterung angepasst.

Häufig kommen mehrschrittige Einstellungsdialoge, sogenannte *Wizards*, zum Einsatz. Diese modernen Dialoge erlauben es, den Inhalt nachfolgender Einstellungsseiten von der vorherigen

Auswahl abhängig zu machen. Die unterste Zeile im Dialog enthält Steuerelemente zur Steuerung der Seitenabfolge. Von einem angebotenen Schritt kann durch Druck auf einen *Weiter*-Button zum nächsten Schritt übergegangen werden, ein Druck auf *Zurück* macht es möglich, alte Einstellungen zu revidieren. Erst ein Klick auf den Button *Fertig* macht die Einstellungen gültig, während ein Klick auf *Abbruch* alle Einstellungen verwirft. Die Einstellungen einer Seite können nur dann verabschiedet werden, wenn sie erfolgreich einer Gültigkeitsprüfung unterworfen wurden, anderenfalls werden die Buttons *Weiter* und *Fertig* gesperrt, ein Informationsfeld im oberen Dialogbereich weist auf die Eingabefehler hin.

Einstellungsmöglichkeiten für Start- und Werkzeugkonfigurationen betten sich, entsprechend den Eclipse-Vorgaben, in einen gemeinsamen Dialog ein. Auf der linken Seite bietet eine Liste mit Steuerelementen die Möglichkeit, Konfigurationen nach Konfigurationstyp getrennt auszuwählen, zu erstellen, zu löschen sowie zu duplizieren. Im unteren Bereich befinden sich zwei Buttons zum Ausführen der ausgewählten Konfiguration und zum Schließen des Dialogs. Die jeweils ausgewählte Konfiguration präsentiert im rechten Bereich des Dialogs entsprechende Steuerelemente, die nach der Art des zu Konfigurierenden auf mehrere Registerkarten aufgeteilt werden können. Hauptelement jeder Konfiguration ist ein oberhalb der Registerkarten angeordnetes Editierfeld zur Änderung des Konfigurationsnamens. Unterhalb der Registerkarten sind zwei Buttons zum Speichern und Revidieren von Änderungen an der ausgewählten Konfiguration.

Zu /F0010/ Konfigurieren der Standardwerkzeuge beim ersten Start

Die Einstellungsmöglichkeiten lassen sich am Besten in einem Wizarddialog in Seiten strukturiert präsentieren; Seite 1 (Abbildung A.3) begrüßt den Nutzer, erklärt ihm die Ursache für das Erscheinen des Dialogs und erklärt ihm, was zu tun ist. Seite 2 (Abbildung A.4) enthält eine Auflistung aller einzustellenden Werkzeuge, wobei je nach Art des Tools ein Editierfeld für den akzeptierten Pfad zur ausführbaren Datei samt einem Schalter mit einem verknüpften Suchdialog hinzugefügt wird.

Zu /F0020/ Manuelles, nachträgliches Konfigurieren der Standardwerkzeuge

Das Konfigurieren der Standardwerkzeuge kann zu jedem späteren Zeitpunkt manuell erfolgen. Der entsprechende Schalter ist auf der über *Run – Tools* erreichbaren Eclipse-Einstellungsseite plaziert (Abbildung A.5). Der Schalter öffnet den zu /F0010/ beschriebenen Wizarddialog.

Zu /F0030/ Erstellung eines Projektes von einer Vorlage

Dieser Dialog ist in den Wizarddialog *New Project* eingebettet. Der Aufruf kann über die Menüpunktfolge *File – New – Project* erfolgen. dort muss unter der Gruppe *Embedded*

Development das Element *Embedded System Project from Template* ausgewählt werden A.6. Auf der zweiten Seite (Abbildung A.7) muss der Name des Projektes eingegeben werden, optional kann ein besonderes Verzeichnis innerhalb des Workspace gewählt werden, standardmäßig ist dies ein gleichnamiger Ordner im Wurzelverzeichnis des Workspace. Auf der dritten Seite (Abbildung A.8) kann bei aktivierter Checkbox auf der linken Seite eine Vorlage ausgewählt werden. Auf der rechten Seite können Informationen über die aktivierte Schablone eingesehen werden. Die letzte Seite des Dialogs (Abbildung A.9) dient zur Eingabe von Parametern der ausgewählten Vorlage, das Erscheinungsbild ist von der gewählten Vorlage abhängig. Für das Minimalprojekt müssen Pfade zu den Werkzeugen ModelSim und Visual Studio angegeben werden. Eine abschließende Fortschrittsanzeige im unteren Bereich informiert den Nutzer über die Aktivität während der Projektgenerierung (Abbildung A.10).

Zu /F0040/ Editieren von C-, VHDL- oder XML-Code

Implementiert als Editor-View nehmen alle Editoren ihren dafür vorgesehenen Platz ein. Eine eigene Perspektive bietet die Möglichkeit vorhandene Zusatzsichten entsprechend zu platzieren. So bieten alle drei Editorarten eine *Outline* genannte Sicht, die eine schnelle Übersicht über die Dokumentenstruktur und Navigierung in selbiger ermöglicht. Die Standardeinstellung der unterschiedlichen Perspektiven zeigen die Abbildungen A.11, A.12 und A.13.

Zu /F0050/ Einrichtung einer Kosimulationskonfiguration

Die Steuerelemente einer Kosimulationskonfiguration ist in 5 Registerkarten aufgeteilt. Auf der Registerkarte *Hardware* (Abbildung A.14) lassen sich Programmpfad, Arbeitsverzeichnis und Argumente für das die Hardware simulierende Programm festlegen. Entsprechende Schalter helfen, Dateien und Resourcen in den Projekten und im globalen Dateisystem aufzufinden und Variablen einzufügen.

Analog dazu sind die Steuerelemente der Registerkarte *Software* angeordnet, auf ihr lässt sich definieren, welches Programm für die Simulation der Software zuständig ist (Abbildung A.15).

Die Registerkarte *Refresh* bestimmt, ob nach nach Beenden beider Anwendungen Änderungen an Dateien und Verzeichnissen innerhalb des Projektbaums im Workspace aktualisiert werden sollen. Welche Resourcen bzw. welcher Teil des Dateibaums einer Überprüfung unterzogen wird und ob die Untersuchung alle Unterverzeichnisse mit einbeziehen soll, kann näher spezifiziert werden (Abbildung A.16).

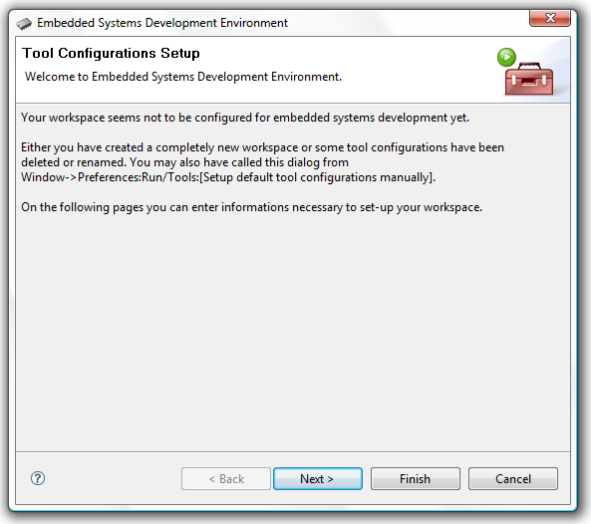


Bild A.3: Seite 1 des Startup-Wizard

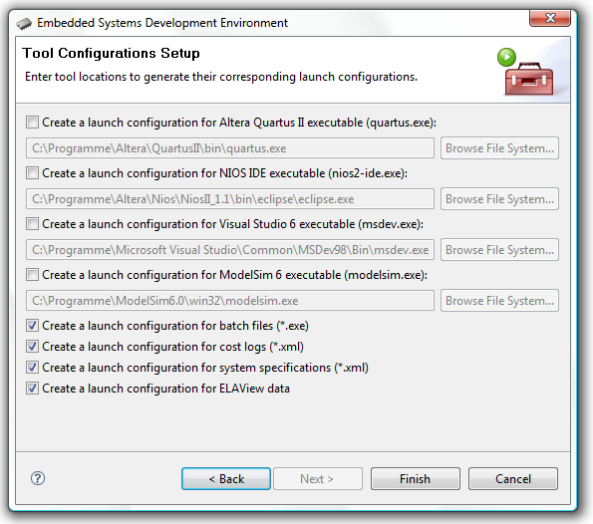


Bild A.4: Seite 2 des Startup-Wizard

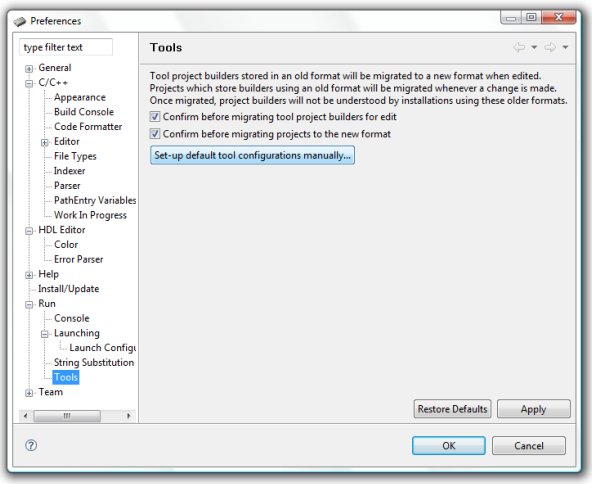


Bild A.5: Startup-Wizard manuell starten

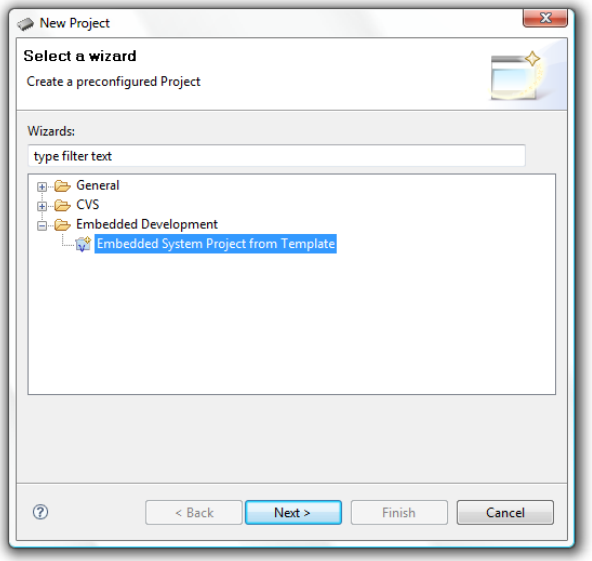


Bild A.6: Seite 1 des New-Project-Wizard

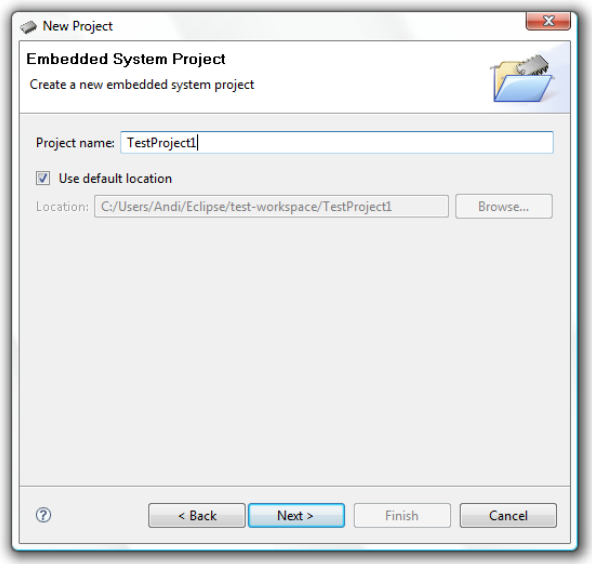


Bild A.7: Seite 2 des New-Project-Wizard

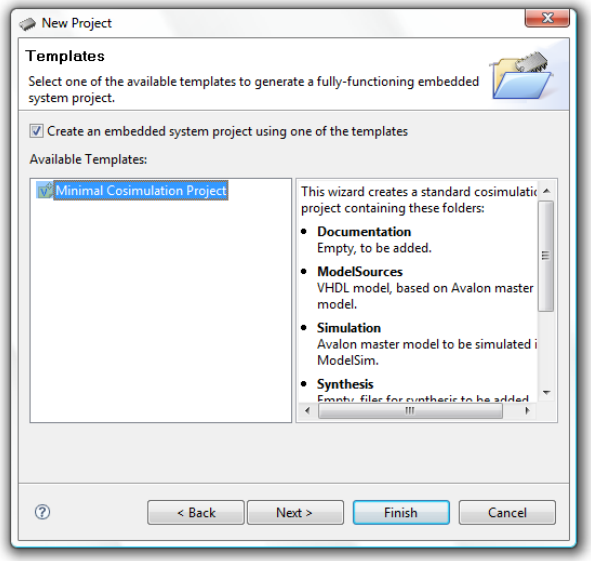


Bild A.8: Seite 3 des New-Project-Wizard

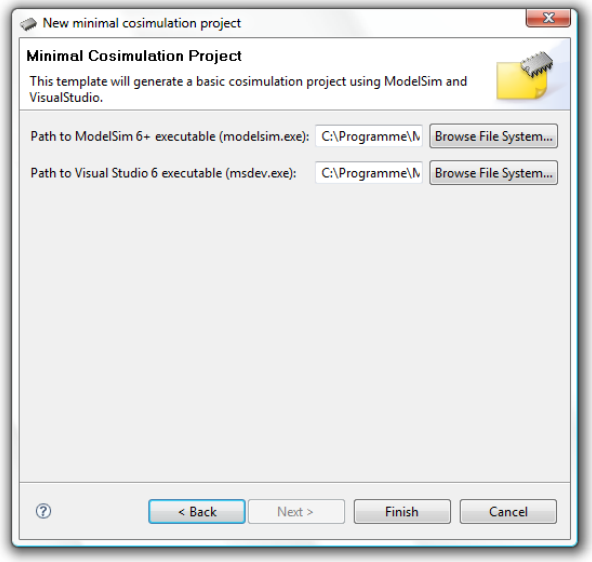


Bild A.9: Seite 4 des New-Project-Wizard

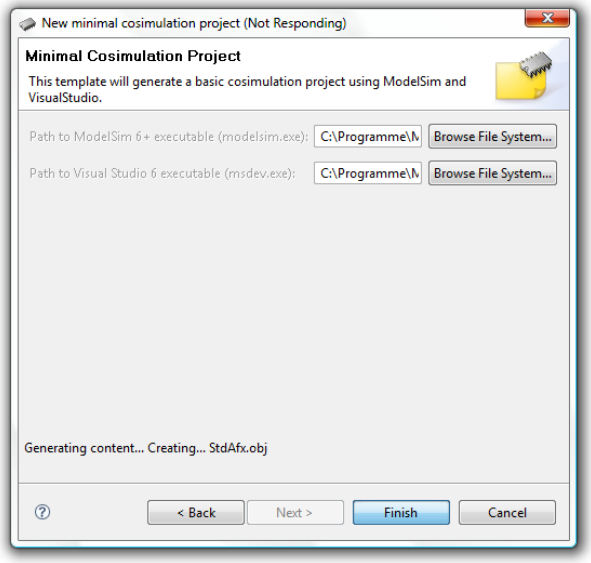


Bild A.10: Fortschrittsanzeige des New-Project-Wizard

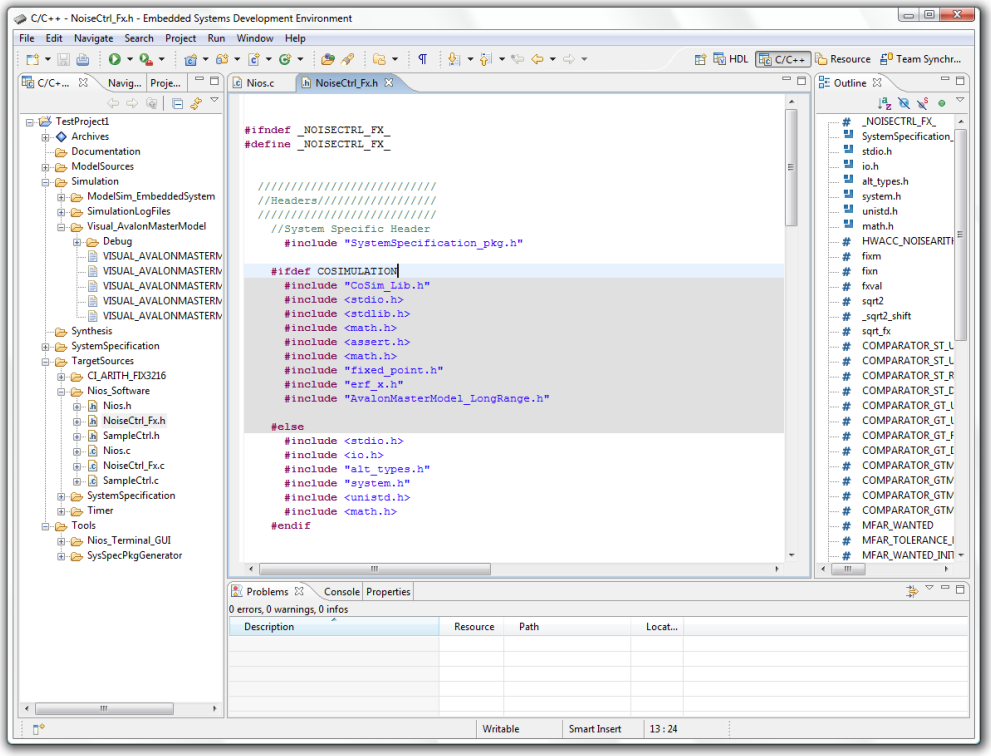


Bild A.11: C/C++-Perspektive

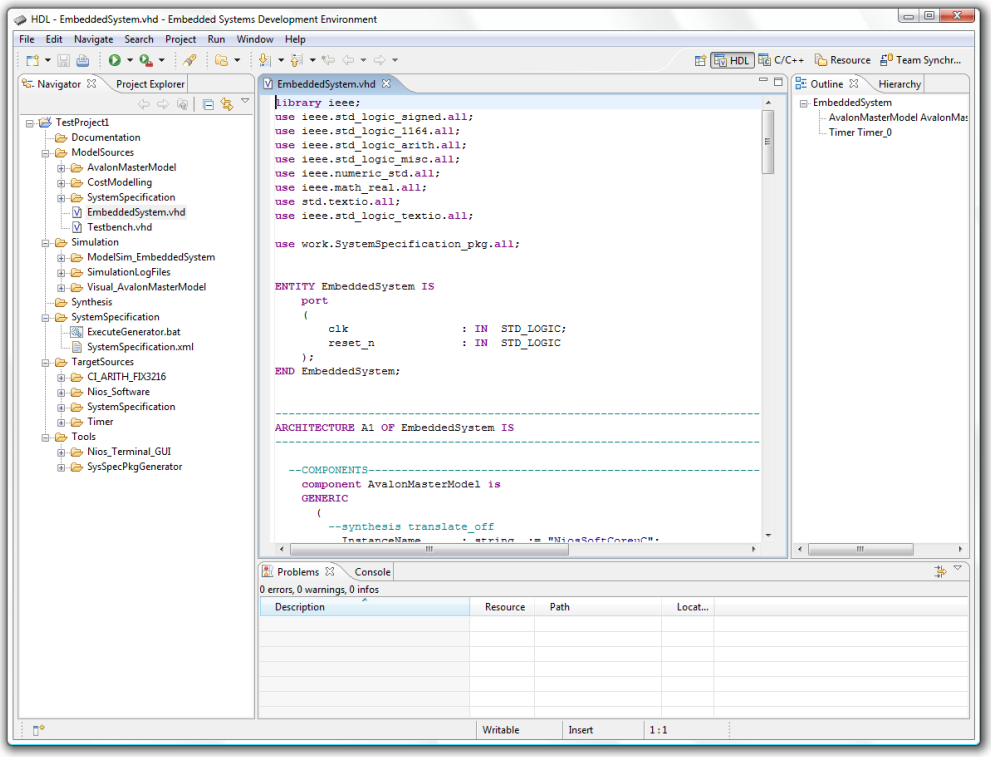


Bild A.12: VHDL-Perspektive

Auf der Registerkarte *Environment* lassen sich für das auszuführende Programm Umgebungsvariablen konfigurieren. Welche Umgebungsvariablen verfügbar sind, zeigt die Liste von Variablennamen-Wert-Paaren. Buttons rechts davon bieten an, neue Umgebungsvariablen hinzuzufügen und vorhandene zu übernehmen. Eine Alternativauswahl im unteren Bereich der Karte überlässt es dem Anwender, ob vom System definierte Variablen sichtbar sind oder nicht (Abbildung A.17).

Bei der Registerkarte *Common* handelt es sich um allgemeine Einstellungen der Konfiguration bezüglich ihres Speicherortes (lokal im Projekt oder global im Workspace sichtbar), und ihres Verhaltens und Erscheinungsbildes im Workspace (Soll die Ausführung unsichtbar erfolgen? Soll die Konfiguration im Favoritenmenü des Workspace erscheinen?). Weiterhin kann festgelegt werden, ob die Standardausgabe des Prozesses überwacht werden soll (Konsole), wie der Text der Standardausgabe kodiert ist, und ob die Standardausgabe in eine Datei oder eine Sicht geschrieben werden soll.

Zu /F0060/ Starten eines Projekts über eine Kosimulationskonfiguration

Das Aufrufen einer Kosimulationskonfiguration kann auf mehrere Arten erfolgen. Entweder kann der Kosimulationskonfigurationsdialog über das Kontextmenü in der Navigationssicht (*Navigator*) oder über die Menüleiste *Run – Run...* geöffnet werden, und hier die entsprechende Kosimulation ausgewählt und der Button *Run* gedrückt werden. Ist die Konfiguration als Favorit gekennzeichnet (siehe Registerkarte *Common* zu /F0050/), ist eine schnellere Anwahl über das Kontextmenü *Run* möglich oder über das Pulldownmenü des Werkzeugicons in der Werkzeugleiste. Ein Aufruf der zuletzt aufgerufenen Konfigurationen ist auch möglich (Abbildung A.19), ein Druck auf das Tool-Icon in der Werkzeugleiste startet die zuletzt ausgeführte Konfiguration.

Zu /F0070/ Konfigurierung eines externen Programms

Dieser Konfigurationstyp befindet sich innerhalb des Dialogs für Werkzeugkonfigurationen. Er besitzt die vier Registerkarten *Main*, *Environment*, *Refresh* und *Common*. Die Registerkarte *Main* dient dazu das auszuführende Programm anzugeben und ist identisch zur Registerkarte Hardware zu /F0050/ gestaltet, alle anderen Registerkarten sind sogar namentlich identisch mit den entsprechenden Registerkarten zu /F0050/ (Abbildungen A.20 - A.23).

Zu /F0080/ Start eines externen Programms

Konfigurationen externer Programme gehören zur Gruppe der Werkzeugkonfigurationen. Wie auch bei Startkonfigurationen lassen sie sich auf unterschiedliche Arten anwenden. Erwähnt sei der Start über die Menüleiste *Run – Tools...* oder das Toolbar-Icon

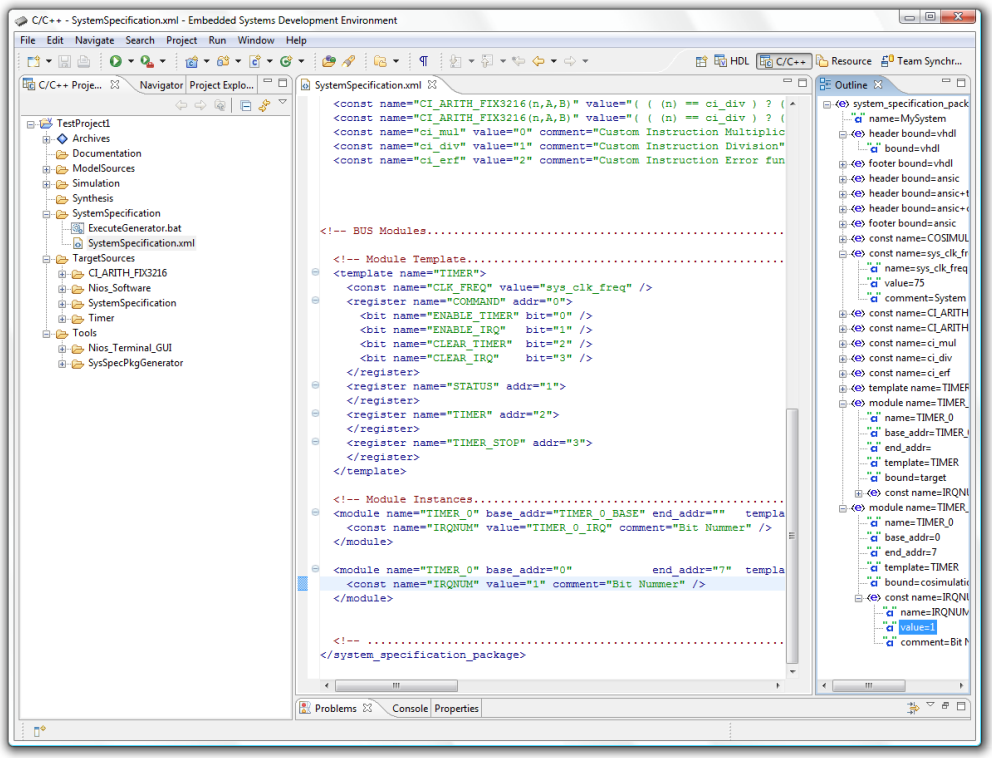


Bild A.13: XML-Perspektive

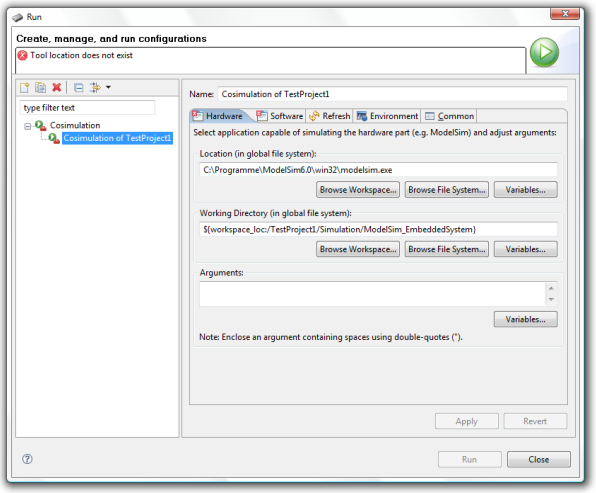


Bild A.14: Registerkarte 1 einer Kosimulationskonfiguration

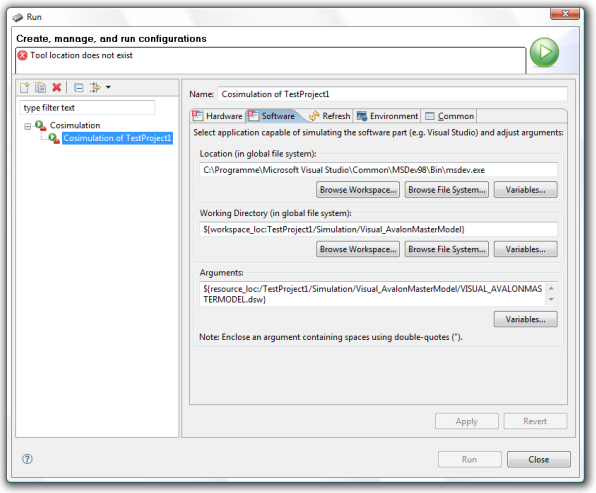


Bild A.15: Registerkarte 2 einer Kosimulationskonfiguration

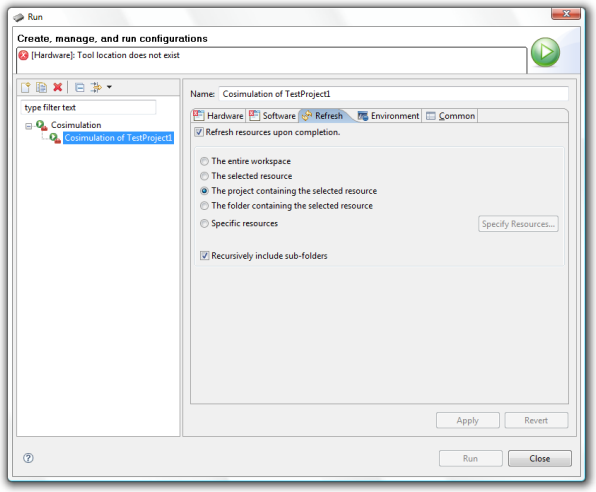


Bild A.16: Registerkarte 3 einer Kosimulationskonfiguration

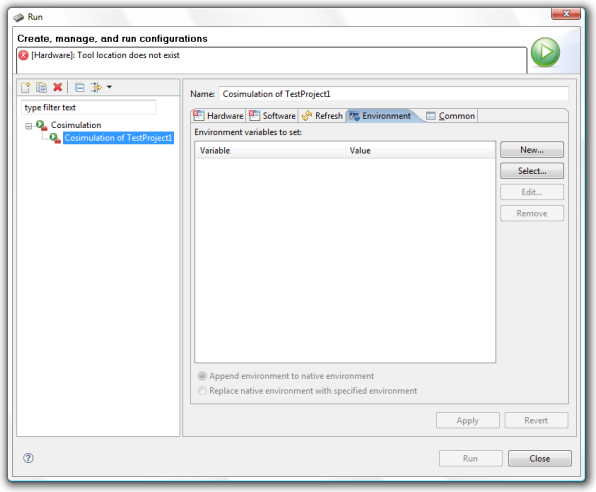


Bild A.17: Registerkarte 4 einer Kosimulationskonfiguration

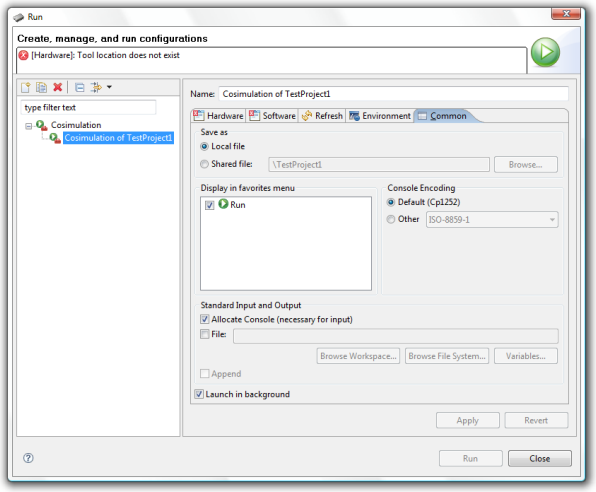


Bild A.18: Registerkarte 5 einer Kosimulationskonfiguration

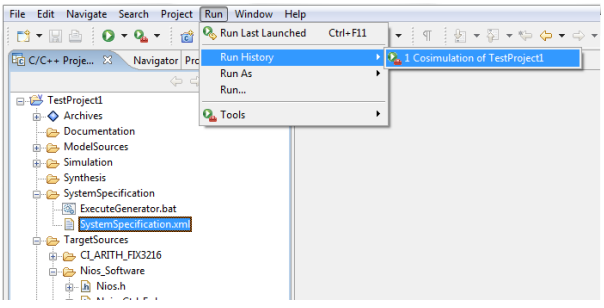


Bild A.19: Start einer Kosimulationskonfiguration

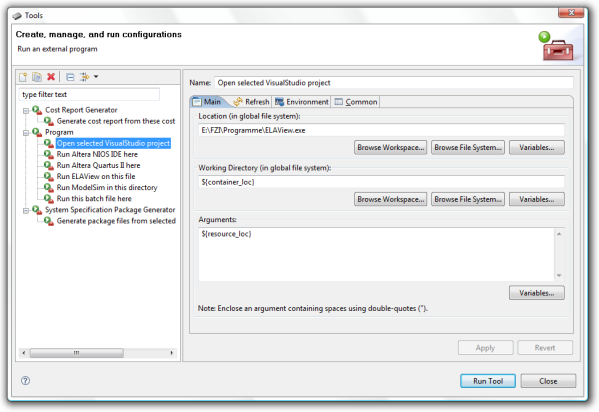


Bild A.20: Registerkarte 1 einer Programmkonfiguration

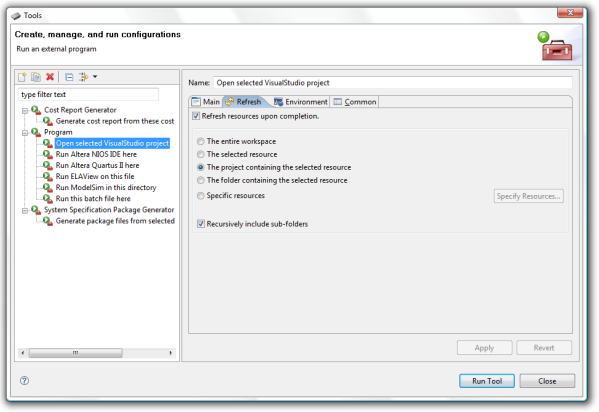


Bild A.21: Registerkarte 2 einer Programmkonfiguration

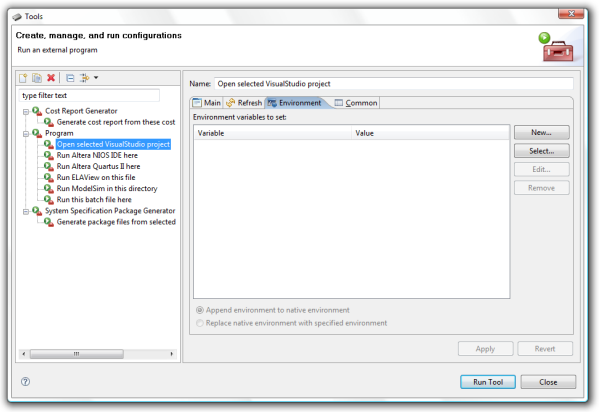


Bild A.22: Registerkarte 3 einer Programmkonfiguration

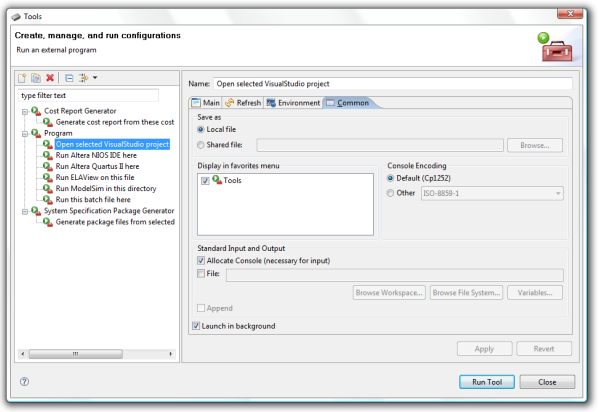


Bild A.23: Registerkarte 4 einer Programmkonfiguration

falls das Werkzeug als Favorit markiert ist. Da oft die Auswahl der Resourcen zum Zeitpunkt der Anwendung des Werkzeugs wichtig ist, bietet sich der Start innerhalb der Resourcensicht (*Navigator View*) an, indem auf die entsprechende Auswahl mit der Maus ein Rechtsklick ausgeführt wird; im sich öffnenden Kontextmenü im Unterpunkt *Apply Tool...* ist die schnelle Auswahl von Favoriten ebenfalls möglich. Bei nicht als Favorit markierten Konfigurationen muss der Einstellungsdialog für Werkzeugkonfigurationen geöffnet werden, dort die entsprechende Konfiguration ausgewählt und *Run Tool* gewählt werden.

Zu /F0090/ Konfigurierung des Generators für Berichte

Die Registerkarte *Main* enthält Editierfelder für die einzulesende XML-Datei und den Ausgabeordner. Beide enthalten einen Button, der einen Auswahldialog von Workspace-Resourcen erlaubt, sowie einen Button um bekannte Variablen einzufügen. Checkbox-Elemente ermöglichen die Wahl, ob PDF, HTML, oder beides generiert werden soll (Abbildung A.24).

Die Registerkarte *Common* enthält allgemeinere Einstellmöglichkeiten, ihr Aufbau entspricht dem in /F0050/ ohne die für die Konsole notwendigen Einstellungen. Im oberen Abschnitt der Karte lässt sich der Speicherort der Konfiguraiton definieren, während im unteren Bereich die Konfiguration im Favoritenmenü sichtbar gemacht werden kann.

Zu /F0100/ Erstellung eines Berichts an Hand von Simulationsdaten

Die Interaktion mit dem System gestaltet sich zunächst analog zum für /F0080/ geschilderten Aufruf (Abbildung A.25). Anschließend gilt es jedoch, in einem sich öffnenden Fenster die Auswahl der erkannten gültigen Kostenberichte anzupassen (Abbildung A.26). Dies ist nötig, da bei selektierten Verzeichnissen alle rekursiv enthaltenen Resourcen überprüft werden. Nach der Generierung öffnen sich die generierten Berichte automatisch in der Editorsicht (Abbildung A.27).

Zu /F0110/ Konfigurierung des Generators für Systemspezifikationen

Die Registerkarte *Main* enthält Editierfelder für die einzulesende XML-Datei und den Ausgabeordner. Beide enthalten einen Button, der einen Auswahldialog von Workspace-Ressourcen erlaubt, sowie einen Button um bekannte Variablen einzufügen. Checkbox-Elemente ermöglichen die Wahl, ob für VHDL, ANSI-C, oder beides generiert werden soll. Bei ausgewähltem Element besteht vorab die Möglichkeit, für die jeweilige Ausgabe zu verwendende Profile kommasepariert anzugeben (Abbildung A.28).

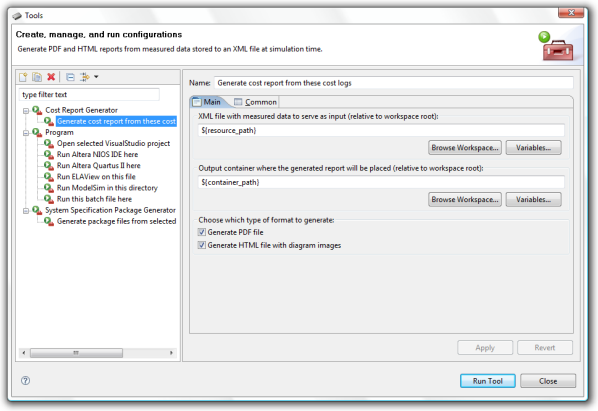


Bild A.24: Einstellung einer Berichtskonfiguration

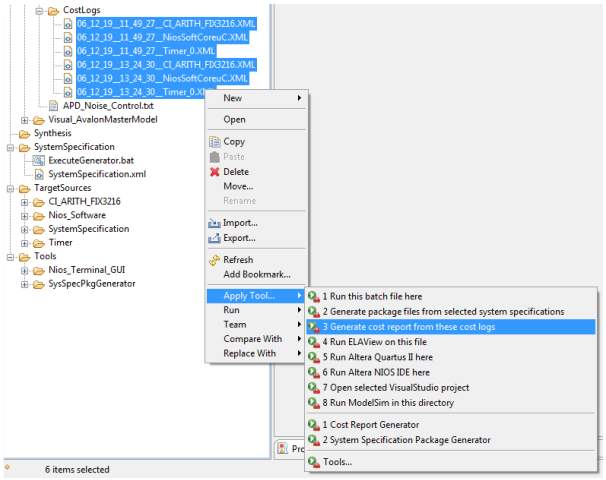


Bild A.25: Schritt 1 beim Erstellen eines Berichts

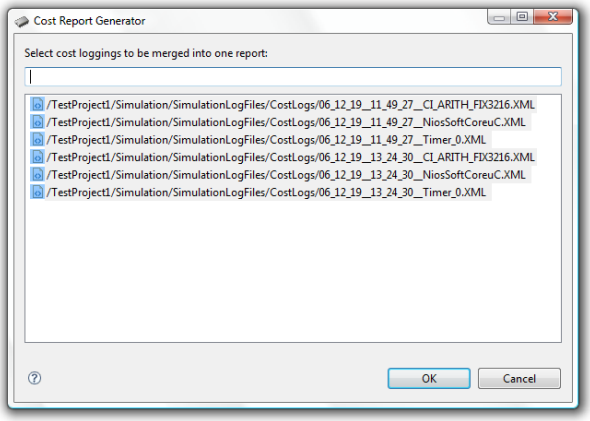


Bild A.26: Schritt 2 beim Erstellen eines Berichts

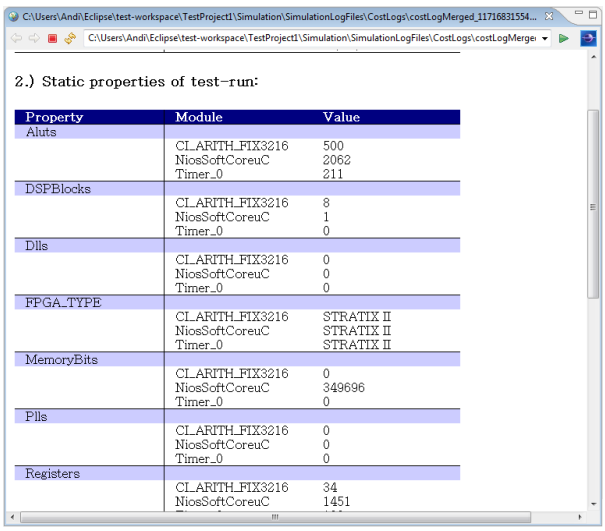


Bild A.27: Erstellten Bericht anzeigen

Die Registerkarte *Common* enthält allgemeinere Einstellmöglichkeiten, ihr Aufbau gleicht dem in /F0050/ beschriebenen ohne die für die Konsole notwendigen Einstellungen. Im oberen Abschnitt der Karte lässt sich der Speicherort der Konfiguraiton definieren, während im unteren Bereich die Konfiguration im Favoritenmenü sichtbar gemacht werden kann.

Zu /F0120/ Erstellung einer Systemspezifikation

Auch die Interaktion zu /F0120/ gestaltet sich analog zum für /F0080/ geschilderten Aufruf (Abbildung A.29). Für VHDL-Ausgabe und ANSI-C-Ausgabe öffnet sich jeweils ein Dialog mit der Möglichkeit die in der Konfiguration definierten Profile anzupassen (Abbildungen A.30 und A.31). Um ein Element hinzuzufügen oder zu entfernen muss bei gedrückter Steuerungstaste (*Strg* bzw. *Ctrl*) mit der Maus ein Linksklick auf das entsprechende Element erfolgen.

A.8 Nichtfunktionale Anforderungen

/NF0010/ Das Erscheinungsbild der Benutzungsoberfläche entspricht den von der Eclipse Foundation gesetzten Regeln für Benutzungsoberflächen, die auf deren Webseite verfügbar sind. Alle Dialoge und Steuerelemente fügen sich in das Erscheinungsbild von Eclipse ein.

/NF0020/ Weitere Symbole (Icons) sollen im Design der Eclipse IDE erstellt werden und im verwendeten Kontext soweit möglich die Identifikation von Daten und Funktionen erleichtern.

Ein wichtiger Aspekt des Systems ist seine *Erweiterbarkeit*. Aus der Sicht des Erweiterers müssen die Geschäftsprozesse in Abbildung A.32 durchführbar sein. Aus diesen Geschäftsprozessen ergeben sich die nachfolgend dargestellten Prozesse bezüglich der Erweiterbarkeit des Systems.

/NF1010/ Neues Werkzeug implementieren

Ziel Ein neuer Konfigurationstyp ist zu implementieren.

Vorbedingung Der einzubindende Programmcode ist in Java geschrieben verfügbar.

Nachbedingung Erfolg Ein neuer Typname innerhalb des Einstellungsdialogs für Toolkonfigurationen im linken Bereich fasst Instanzen seiner selbst zusammen. Diese Instanzen besitzen eine Reihe von speziell angepassten und auf der rechten Seite dargestellten Einstellungen, als Registerkarten dargestellt.

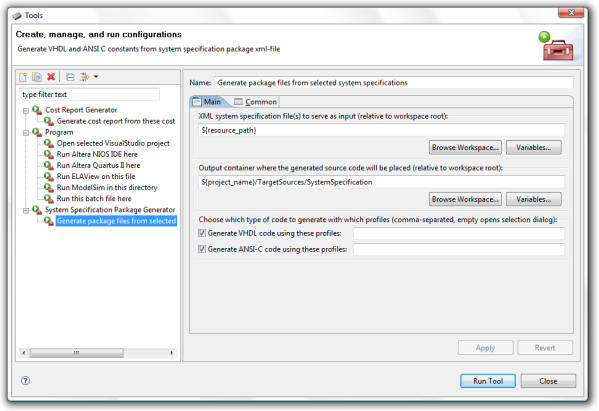


Bild A.28: Einstellung einer Systemspezifikationskonfiguration

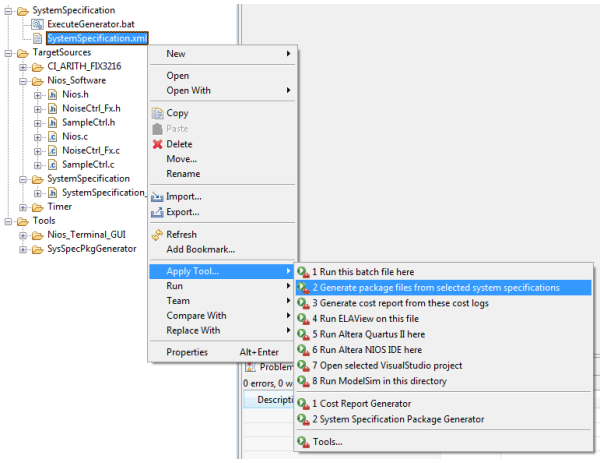


Bild A.29: Schritt 1 beim Generieren einer Systemspezifikation

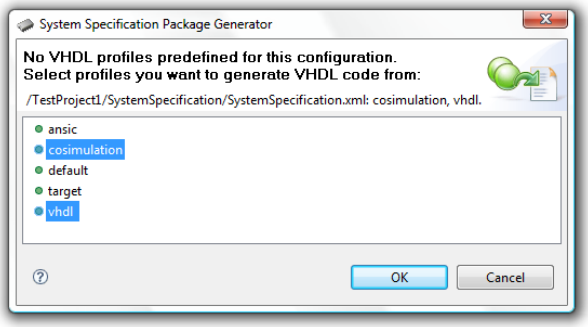


Bild A.30: Schritt 2 beim Erstellen einer Systemspezifikation

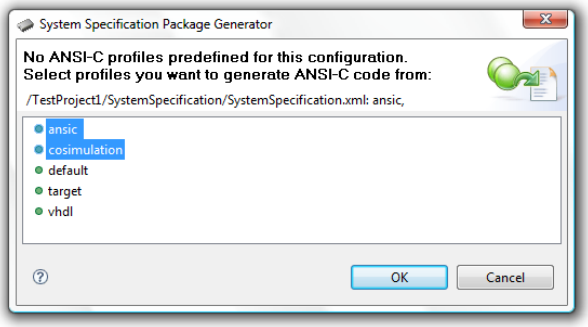


Bild A.31: Schritt 3 beim Erstellen einer Systemspezifikation

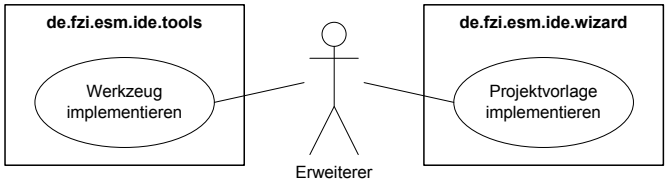


Bild A.32: Geschäftsprozesse des Erweiterers

Nachbedingung Fehlschlag Im Einstellungsdialog für Konfigurationen befindet sich kein neuer Toolkonfigurationstyp.

Auslösendes Ereignis Für ein Projekt wird ein neuer Werkzeugtypus benötigt.

Voreinstellung Es existieren bereits drei Arten von Werkzeugkonfigurationen: *Cost Report Generator*, *System Specification Package Generator*, *Program*.

Beschreibung Eigene Implementierungen von Werkzeugen können als externes ausführbares Programm in das System eingebunden werden. Eine weitere Möglichkeit besteht darin, den Quellcode direkt als Pluginerweiterung einzubinden, um von den Möglichkeiten der IDE profitieren zu können. Der Vorteil besteht in der direkteren Einbettung in die Eclipseumgebung und das Standard Widget Toolkit (SWT), und der direkten Verfügbarkeit des Werkzeugs ohne umständliche Anbindung als externes Programm.

/NF1020/ Neue Projektvorlage erstellen

Ziel Eine neue Projektvorlage ist einzubinden.

Vorbedingung Das als Vorlage einzubindende Projekt existiert bereits.

Nachbedingung Erfolg Über *File – New – Project...* lässt sich bei Auswahl von *Embedded System Project from Template* auf der zweiten Einstellungsseite die Vorlage auswählen und auf der nachfolgenden Seite parametrisieren.

Nachbedingung Fehlschlag Die Projektschablone ist im Dialog nicht vorhanden.

Auslösendes Ereignis Es existiert ein Projekt, das als Vorlage verwendbar sein soll.

Voreinstellung Eine minimale Projektschablone *Minimal Cosimulation Project* ist bereits integriert.

Beschreibung Der größte Nutzen einer Projektschablone besteht in der Zeitersparnis. Eigene Projektschablonen erleichtern das häufige Neuerstellen von Projekten, die schon gewisse oft vorausgesetzte Ausprägungen mitbringen sollen.

A.9 Technische Produktumgebung

Das System selbst stellt gewisse Erwartungen an seine Umgebung, die es einzuhalten gilt.

A.9.1 Software

- Java Laufzeitumgebung Version 1.5 mindestens in der Standardausgabe (J2SE 5)
- Betriebssystem für das die genannte Java-Plattform verfügbar ist

A.9.2 Hardware

- Intel Pentium 4 Prozessor mit mindestens 2,4 GHz Taktfrequenz
- 1024 MB Arbeitsspeicher
- LAN-Anschluss mit Verbindung ins WAN
- möglichst ein 19" TFT-Monitor mit einer Auflösung von 1280×1024
- Tastatur, 3-Tasten-Maus mit Scrollrad

A.9.3 Orgware

- garantierte Aktualität des Betriebssystems
- Kompatibilität der Umgebung mit der Anwendung, das heißt die Hardware, die Version des Betriebssystems, der JVM und die Version von Eclipse müssen mit dem Produkt verträglich sein.

A.9.4 Produkt-Schnittstellen

Diese vier Anwendungen müssen für die Arbeit mit dem System installiert sein. Um eine optimale Eingliederung des Systems in den Entwurfsprozess zu erhalten, sind insbesondere die Versionen der Anwendungen zu beachten.

- MentorGraphics ModelSim 6
- Microsoft Visual Studio 6
- Altera Quartus II
- Altera NIOS II IDE

A.10 Spezielle Anforderungen an die Entwicklungsumgebung

Zur Entwicklung des Systems müssen besondere Anforderungen an Hardware, Software und Orgware erfüllt sein. Teilweise hat der Kunde für deren Einhaltung zu sorgen.

A.10.1 Software

- Eclipse 3.2
- Java Laufzeitumgebung Version 1.5 mindestens in der Standardausgabe (J2SE 5)
- Windows XP oder Windows Vista
- UltraEdit Texteditor
- Altova XMLSpy Enterprise Edition Version 2007 SP1
- MikTeX ab der Version 2.5 und TeXnicCenter ab der Version 1 Beta 7.01

A.10.2 Hardware

- Intel Pentium 4 Prozessor mit mindestens 2,4 GHz Taktfrequenz
- 1014 MB Arbeitsspeicher
- 4 GB Platz auf der lokalen Festplatte
- 19" TFT-Monitor mit einer Auflösung von 1280×1024
- Tastatur, 3-Tasten-Maus mit Scrollrad

A.10.3 Orgware

- Gewährleistung einer permanenten Internetanbindung
- telefonische Verfügbarkeit des Kunden

A.11 Gliederung in Teilprodukte

Das Produkt lässt sich in folgende Teilprodukte zerlegen, die voneinander unabhängig sind. Die Entwicklung erfolgt sequentiell in dieser Reihenfolge:

1. Template Wizard
2. Werkzeuganbindung und -verbindung
3. Auswertung der Simulationsdaten

A.12 Ergänzungen

- Das System kann um ein kontextsensitives Hilfesystem erweitert werden. Jeder Dialog enthält einen Button für den Aufruf einer Hilfeseite.
- Dem System kann eine Willkommensseite mit einer Einführung, Tutorials, Beispielen etc. hinzugefügt werden.

B Installation

In diesem Kapitel werden alle Schritte beschrieben, die zu einer an den Kunden verteilbaren Version führen. Einzige Voraussetzung ist ein Rechner, auf dem *Windows XP* oder *Windows Vista* läuft. Sämtliche zur Erstellung des Produkts notwendigen Daten sind im Plugin *de.fzi.esm.ide* zusammengefasst.

B.1 Eclipse einrichten

Diese Schritte richten Eclipse samt Arbeitsbereich ein:

1. Installation von Eclipse 3.2.2 [Deu07a]
2. Installation der BIRT 2.1.2 Komponenten:
   a) Eclipse starten, einen neuen Ordner als Arbeitsbereich auswählen.
   b) Über das Menü *Help – Software Updates – Find And Install...* den Aktualisierungsdialog öffnen. *Search for new features to install* auswählen, das Häkchen bei *Callisto Discovery Site* setzen, auf *Finish* klicken.
   c) In der daraufhin erscheinenden Auswahl *Callisto Discovery Site – Charting and Reporting – Eclipse BIRT Report Designer Framework 2.1.2.v???* wählen, den Button *Select Required* anklicken.
   d) Um mit Birt PDF-Dateien generieren zu können, muss die mitgelieferte Komponente `itext-1.3.jar` in den Unterordner

      `\plugin\com.lowagie.itext_???\lib`

      des Eclipse-Verzeichnisses kopiert werden.
3. Alle systemeigenen Plugins aus dem Archiv `de.fzi.esm.ide (????.??.??).zip` in den Arbeitsbereich über den Menüpunkt *Run – File – Import* importieren. Alle hinzugefügten Projekte sollten ohne Fehler kompilieren.

B.2 Deployment des Systems

Mit diesen Schritten kann eine Version erstellt werden, die an den Kunden weitergegeben werden kann:

1. Den Menüpunkt *File – Export...* auswählen.
2. Im erscheinenden Dialog *Eclipse product* selektieren, *Next* drücken.
3. Als Konfiguration `/de.fzi.esm.ide/ide.product` auswählen, das Wurzelverzeichnis ist beliebig, kann aber zum Beispiel `esde` lauten. Das Häkchen bei *Synchronize before exporting* gesetzt lassen. Als Ziel ein beliebiges Verzeichnis wählen, beispielsweise könnte dies

   ```
   C:\Dokumente und Einstellungen\<Username>\Desktop
   ```

   sein. Die Quelltexte nicht zum exportieren Produkt hinzufügen. Auf den Button *Finish* klicken.

Das erstellte Generat sollte selbständig lauffähig sein.

C Kostenbericht

In Kapitel 3.3.1 wurde über die Realisierung des Generators für Kostenberichte gesprochen. Dieses Werkzeug ermöglicht das automatische Generieren von Kostenberichten aus in der Simulation gemessenen Daten. Nachfolgend ist ein solcher Bericht dargestellt, der die vergleichende Bewertung unterschiedlicher Versionen einer Architektur eines Hardware-/Software-Systems zulässt. Es wird die ursprüngliche Version der Architektur der optimierten Version gegenübergestellt. Der untere Graph mit dem Namen »Average Avalon Master Peripheral Data Bus Usage« (auf Seite 22 des Berichts) zeigt, dass die modifizierte VHDL-Konfiguration – im Bericht *SC2* genannt – eine niedrigere Buslast verursacht und damit eine Optimierung darstellt.

Co-Simulation Cost Report

Cost calculation rendered during a ESDE run in a cost-simulated environment.

Created from: D:\FZI_FARO\Development\FPGA\ESDE_Workspace\.metadata\.plugins\de.fzi.esm.ide.tools\costLogMerged_1175701609956.xml

Created on: 04.04.2007 17:46

Comparing: 2 Systems

1.) Involved Systems Configuration

Module Name	Configuration	Simulation Time
ADC_CTRL_1		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10
ADC_CTRL_2		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10
AcamCtrl		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10
CI_ARITH_FIX3216		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10
LongRangeBuffer		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10
LongRangeNoiseArith		
SC1	A1	03/29/07 10:31:31
NIOS		
SC1	A4	03/29/07 10:31:31
SC2	A4	03/29/07 10:45:10
NoiseArith FX on NIOS		
SC1	HW ACC	03/29/07 10:31:31
SC2	SW	03/29/07 10:45:10
NoiseThresholdCtrl		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Module Name	Configuration	Simulation Time
PulsePrediction		
SC1	A1	03/29/07 10:31:31
SC2	A1	03/29/07 10:45:10

Copyright © 2006 FZI ESS (www.fzi.de/ess)

2.) Static System Parameters

Aluts	Module Name	Value
	ADC_CTRL_1	
	SC1	2199
	SC2	2199
	ADC_CTRL_2	
	SC1	2199
	SC2	2199
	AcamCtrl	
	SC1	166
	SC2	166
	CI_ARITH_FIX3216	
	SC1	500
	SC2	500
	LongRangeBuffer	
	SC1	222
	SC2	222
	NIOS	
	SC1	2062
	SC2	2062
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	862
	SC2	862
	PulsePrediction	
	SC1	773
	SC2	773

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

	PulsePrediction	
	LongRangeNoiseArith	
	SC1	4943

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

Aluts	System Configuration	Value
∑		
	SC1	13926
	SC2	8983

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

DSPBlocks	Module Name	Value
	ADC_CTRL_1	
	SC1	0
	SC2	0
	ADC_CTRL_2	
	SC1	0
	SC2	0
	AcamCtrl	
	SC1	0
	SC2	0
	CI_ARITH_FIX3216	
	SC1	8
	SC2	8
	LongRangeBuffer	
	SC1	0
	SC2	0
	NIOS	
	SC1	1
	SC2	1
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	0
	SC2	0
	PulsePrediction	
	SC1	0
	SC2	0
	LongRangeNoiseArith	
	SC1	16

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

DSPBlocks	System Configuration	Value
Σ		
	SC1	25
	SC2	9

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

Dlls	Module Name	Value
	ADC_CTRL_1	
	SC1	0
	SC2	0
	ADC_CTRL_2	
	SC1	0
	SC2	0
	AcamCtrl	
	SC1	0
	SC2	0
	CI_ARITH_FIX3216	
	SC1	0
	SC2	0
	LongRangeBuffer	
	SC1	0
	SC2	0
	NIOS	
	SC1	0
	SC2	0
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	0
	SC2	0
	PulsePrediction	
	SC1	0
	SC2	0
	LongRangeNoiseArith	
	SC1	0

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

Dlls	System Configuration	Value
Σ		
	SC1	0
	SC2	0

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

FPGA_TYPE	Module Name	Value
	ADC_CTRL_1	
	SC1	STRATIX II
	SC2	STRATIX II
	ADC_CTRL_2	
	SC1	STRATIX II
	SC2	STRATIX II
	AcamCtrl	
	SC1	STRATIX II
	SC2	STRATIX II
	CI_ARITH_FIX3216	
	SC1	STRATIX II
	SC2	STRATIX II
	LongRangeBuffer	
	SC1	STRATIX II
	SC2	STRATIX II
	NIOS	
	SC1	STRATIX II
	SC2	STRATIX II
	NoiseArith FX on NIOS	
	SC1	STRATIX_II
	SC2	STRATIX_II
	NoiseThresholdCtrl	
	SC1	STRATIX II
	SC2	STRATIX II
	PulsePrediction	
	SC1	STRATIX II
	SC2	STRATIX II
	LongRangeNoiseArith	
	SC1	STRATIX II

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

MemoryBits	Module Name	Value
	ADC_CTRL_1	
	SC1	0
	SC2	0
	ADC_CTRL_2	
	SC1	0
	SC2	0
	AcamCtrl	
	SC1	0
	SC2	0
	CI_ARITH_FIX3216	
	SC1	0
	SC2	0
	LongRangeBuffer	
	SC1	8192
	SC2	8192
	NIOS	
	SC1	349696
	SC2	349696
	NoiseArith FX on NIOS	
	SC1	3000
	SC2	3000
	NoiseThresholdCtrl	
	SC1	0
	SC2	0
	PulsePrediction	
	SC1	4096
	SC2	4096
	LongRangeNoiseArith	
	SC1	422

Copyright © 2006 FZI ESS (www.fzi.de/ess)

MemoryBits	System Configuration	Value
∑		
	SC1	365406
	SC2	364984

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Plls	Module Name	Value
	ADC_CTRL_1	
	SC1	0
	SC2	0
	ADC_CTRL_2	
	SC1	0
	SC2	0
	AcamCtrl	
	SC1	0
	SC2	0
	CI_ARITH_FIX3216	
	SC1	0
	SC2	0
	LongRangeBuffer	
	SC1	0
	SC2	0
	NIOS	
	SC1	0
	SC2	0
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	0
	SC2	0
	PulsePrediction	
	SC1	0
	SC2	0
	LongRangeNoiseArith	
	SC1	0

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Plls	System Configuration	Value
∑		
	SC1	0
	SC2	0

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Registers	Module Name	Value
	ADC_CTRL_1	
	SC1	1817
	SC2	1817
	ADC_CTRL_2	
	SC1	1817
	SC2	1817
	AcamCtrl	
	SC1	141
	SC2	141
	CI_ARITH_FIX3216	
	SC1	34
	SC2	34
	LongRangeBuffer	
	SC1	208
	SC2	208
	NIOS	
	SC1	1451
	SC2	1451
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	559
	SC2	559
	PulsePrediction	
	SC1	542
	SC2	542
	LongRangeNoiseArith	
	SC1	3343

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

Registers	System Configuration	Value
$\sum$		
	SC1	9912
	SC2	6569

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

fmax	Module Name	Value
	ADC_CTRL_1	
	SC1	250
	SC2	250
	ADC_CTRL_2	
	SC1	250
	SC2	250
	AcamCtrl	
	SC1	145
	SC2	145
	CI_ARITH_FIX3216	
	SC1	111
	SC2	111
	LongRangeBuffer	
	SC1	180
	SC2	180
	NIOS	
	SC1	80
	SC2	80
	NoiseArith FX on NIOS	
	SC1	0
	SC2	0
	NoiseThresholdCtrl	
	SC1	129
	SC2	129
	PulsePrediction	
	SC1	85
	SC2	85
	LongRangeNoiseArith	
	SC1	75

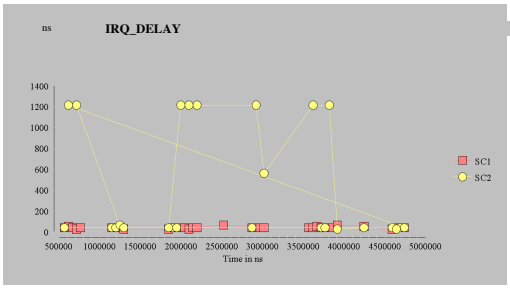
Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

Range (f)	System Configuration	Value
$[f_{min}...f_{max}]$		
	SC1	75 - 250 MHz
	SC2	80 - 250 MHz

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

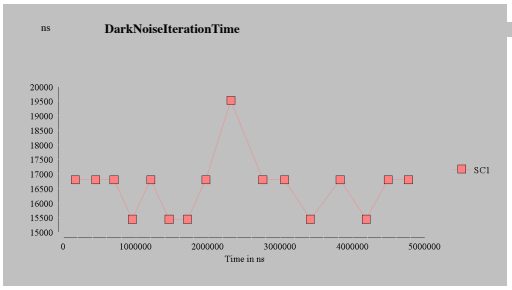
3.) Dynamic System Parameters

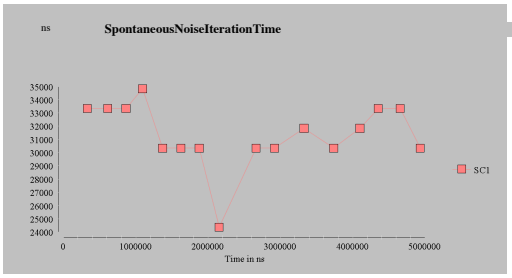
ADC_CTRL_1



Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

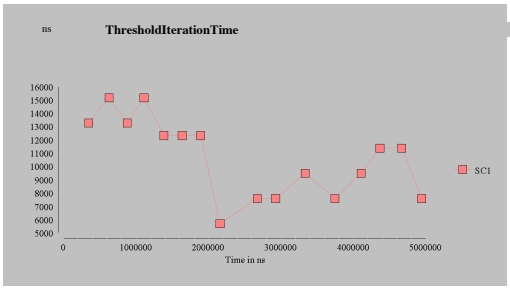
LongRangeNoiseArith





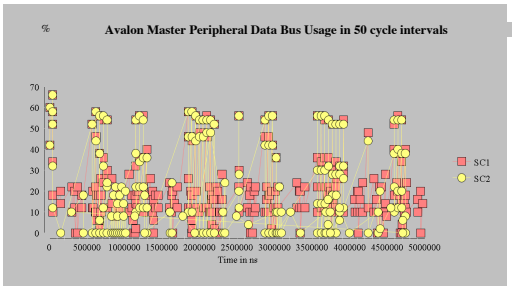
Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

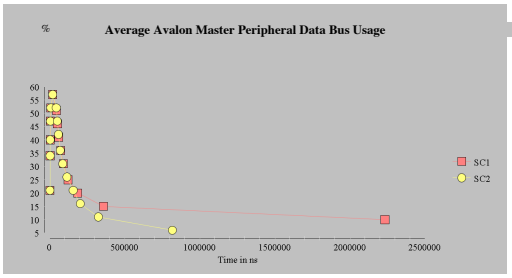
LongRangeNoiseArith



Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

NIOS





Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

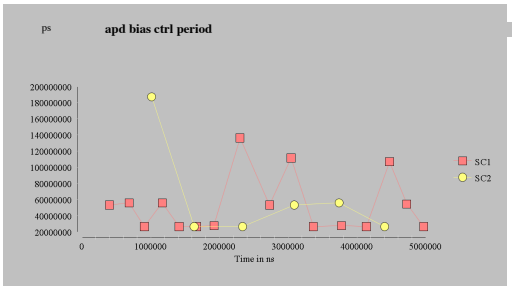
23 / 30

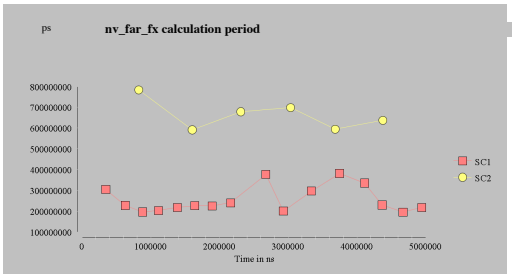
NIOS

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

24 / 30

NoiseArith FX on NIOS





Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

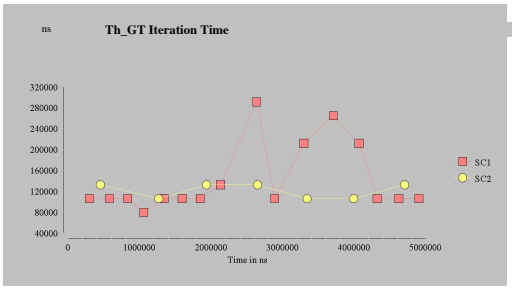
25 / 30

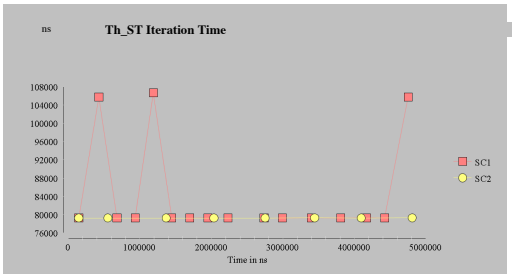
NoiseArith FX on NIOS

Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

26 / 30

NoiseThresholdCtrl



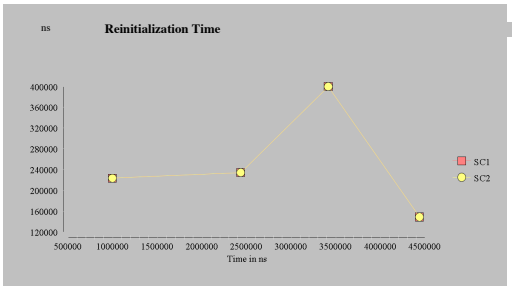


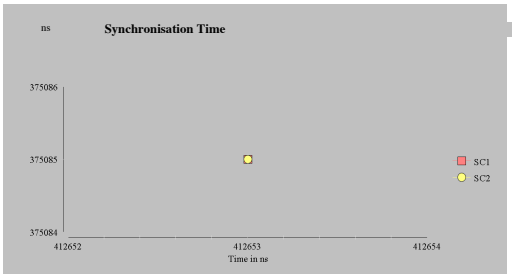
Cost report created on 04.04.2007 17:47
Copyright © 2006 FZI ESS (www.fzi.de/ess)

NoiseThresholdCtrl

Copyright © 2006 FZI ESS (www.fzi.de/ess)

PulsePrediction





Copyright © 2006 FZI ESS (www.fzi.de/ess)

PulsePrediction

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Copyright © 2006 FZI ESS (www.fzi.de/ess)

Die einzulesenden Messdaten eines Moduls ebenso wie die kumulierten Messdaten mehrerer Module liegen in einer XML-Datei vor, die demselben Schema genügen müssen. Diese XSD-Datei dient der Validierung des Datenformats:

Listing C.1: Die XML-Schema-Definition `costLog.xsd` beschreibt die Messdatenstruktur

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="CostLog">
    <xs:complexType>
      <xs:all>
        <xs:element ref="Header"/>
        <xs:element ref="Static_Hw_Costs"/>
        <xs:element ref="Dynamic_Hw_Costs"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="Header">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Module"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Module">
    <xs:complexType>
      <xs:all>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Configuration" type="xs:string"/>
        <xs:element name="Date" type="xs:string"/>
        <xs:element name="Time" type="xs:string"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="Static_Hw_Costs">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="StaticFactor"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="StaticFactor">
    <xs:complexType>
      <xs:all>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Value" type="xs:string"/>
        <xs:element minOccurs="0" name="Unit" type="xs:string"/>
      </xs:all>
      <xs:attribute name="moduleName" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleDate" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleTime" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleConfiguration" type="xs:string"/>
        <!--optional for merging logs-->
    </xs:complexType>
  </xs:element>
```

```
  <xs:element name="Dynamic_Hw_Costs">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="DynamicFactor"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="DynamicFactor">
    <xs:complexType>
      <xs:all>
        <xs:element name="Time" type="xs:string"/>
        <xs:element name="CostName" type="xs:string"/>
        <xs:element minOccurs="0" name="CostValue" type="xs:string"/>
        <xs:element minOccurs="0" name="CostUnit" type="xs:string"/>
        <xs:element minOccurs="0" name="CostTime" type="xs:string"/>
        <!--
          <CostTime>10 ns</CostTime> parsed later to
          <CostValue>10</CostValue><CostUnit>ns</CostUnit>
          Reason: VHDL cost log doesn´t support splitting "10␣ns" into "10" "ns"
        -->
      </xs:all>
      <xs:attribute name="moduleName" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleDate" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleTime" type="xs:string"/> <!--optional for merging logs-->
      <xs:attribute name="moduleConfiguration" type="xs:string"/>
        <!--optional for merging logs-->
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Das Verschmelzen von Messdaten mehrerer Module übernimmt nachfolgendes XSL-Transformationsskript. Dabei geht das Skript so vor, dass immer nur eine XML-Datei in die XML-Datei mit dem kumulierten Inhalt übernommen wird. Welche Datei hinzugefügt wird, ist über den Parameter `targetFile` anzugeben. Hauptaufgabe des Skripts ist, beim Einfügen neuer Daten die Modulinformationen `moduleName`, `moduleDate`, `moduleTime` und `moduleConfiguration` als Attribute an die Daten anzuhängen, da diese Informationen sonst verloren gehen würden.

Listing C.2: Das XSLT-Skript `costLogMerge.xsl` kumuliert die Messdaten zweier Module

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  exclude-result-prefixes="xs␣fn␣xsl"
>
  <!--output format close to xquery-->
  <xsl:output method="xml" encoding="utf-8" omit-xml-declaration="yes"/>
  <xsl:param name="targetFile"/>
  <xsl:variable name="moduleName" select="//CostLog/Header/Module/Name"/>
  <xsl:variable name="moduleDate" select="//CostLog/Header/Module/Date"/>
  <xsl:variable name="moduleTime" select="//CostLog/Header/Module/Time"/>
  <xsl:variable name="moduleConfiguration" select="//CostLog/Header/Module/Configuration"/>
```

```
<xsl:template match="//CostLog">
  <xsl:copy>
    <xsl:apply-templates select="Header"/>
    <xsl:apply-templates select="Static_Hw_Costs"/>
    <xsl:apply-templates select="Dynamic_Hw_Costs"/>
  </xsl:copy>
</xsl:template>
<xsl:template match="//CostLog/Header">
  <xsl:copy>
    <xsl:copy-of select="document($targetFile)//CostLog/Header/*">
      <!--xsl:if test=".!='ModuleName' and not(exists(@moduleName))">
        <xsl:message>CostLog/Header/Date|Time is missing moduleName!</xsl:message>
      </xsl:if-->
    </xsl:copy-of>
    <xsl:copy-of select="Module"/>
    <!--xsl:for-each select="Time|Date|Configuration">
      <xsl:copy>
        <xsl:attribute name="moduleName">
        <xsl:value-of select="$moduleName"/>
        </xsl:attribute>
        <xsl:value-of select="."></xsl:value-of>
      </xsl:copy>
    </xsl:for-each-->
  </xsl:copy>
</xsl:template>
<xsl:template match="//CostLog/Static_Hw_Costs">
  <xsl:copy>
    <xsl:copy-of select="document($targetFile)//CostLog/Static_Hw_Costs/*">
      <xsl:if test=".!='ModuleName' and not(exists(@moduleName))">
        <xsl:message>
          CostLog/Static_Hw_Factor/StaticFactor is missing moduleName!
        </xsl:message>
      </xsl:if>
    </xsl:copy-of>
    <xsl:for-each select="StaticFactor">
      <xsl:copy>
        <xsl:attribute name="moduleName">
          <xsl:value-of select="$moduleName"/>
        </xsl:attribute>
        <xsl:attribute name="moduleDate">
          <xsl:value-of select="$moduleDate"/>
        </xsl:attribute>
        <xsl:attribute name="moduleTime">
          <xsl:value-of select="$moduleTime"/>
        </xsl:attribute>
        <xsl:attribute name="moduleConfiguration">
          <xsl:value-of select="$moduleConfiguration"/>
        </xsl:attribute>
        <xsl:copy-of select="child::*"></xsl:copy-of>
      </xsl:copy>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>
<xsl:template match="//CostLog/Dynamic_Hw_Costs">
  <xsl:copy>
```

```
      <xsl:copy-of select="document($targetFile)/CostLog//Dynamic_Hw_Costs/*">
        <xsl:if test=".!='ModuleName' and not(exists(@moduleName))">
          <xsl:message>
            CostLog/Dynamic_Hw_Factor/DynamicFactor is missing moduleName!
          </xsl:message>
        </xsl:if>
      </xsl:copy-of>
      <xsl:for-each select="DynamicFactor">
        <xsl:copy>
          <xsl:attribute name="moduleName">
            <xsl:value-of select="$moduleName"/>
          </xsl:attribute>
          <xsl:attribute name="moduleDate">
            <xsl:value-of select="$moduleDate"/>
          </xsl:attribute>
          <xsl:attribute name="moduleTime">
            <xsl:value-of select="$moduleTime"/>
          </xsl:attribute>
          <xsl:attribute name="moduleConfiguration">
            <xsl:value-of select="$moduleConfiguration"/>
          </xsl:attribute>
          <xsl:apply-templates mode="resolveTime" select="*"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>

  <xsl:template mode="resolveTime" match="CostTime">
    <CostValue><xsl:value-of select="substring-before(.,' ')"/></CostValue>
    <CostUnit><xsl:value-of select="substring-after(.,' ')"/></CostUnit>
    <!--
      <CostTime>10 ns</CostTime> parsed to <CostValue>10</CostValue><CostUnit>ns</CostUnit>
      Reason: VHDL cost log doesn´t support splitting "10 ns" into "10" "ns"
    -->
  </xsl:template>
  <xsl:template mode="resolveTime" match="*">
    <xsl:copy-of select="."></xsl:copy-of>
  </xsl:template>
</xsl:stylesheet>
```

Glossar

BIRT Eine Erweiterung für die Eclipse-Plattform, die das Auswerten und das anschließende Präsentieren von Daten in einem Bericht ermöglicht. Der Bericht kann in den Formaten PDF und HTML erfolgen, das DOC-Format von Word befindet sich noch in der Entwicklung. Als Datenquellen können XML-Dateien, eine SQL-Datenbank oder eine beliebige andere Datenquelle dienen, die mit Javacode ausgelesen werden kann. Für das Anbinden von Datenquellen und das Editieren und Layouten eines Berichtes wird ein graphischer Editor innerhalb der Eclipse-IDE zur Verfügung gestellt. Die Generierung ist durch *Scripting* anpassbar: Mit Hilfe von Javascript kann an verschiedenen Stellen in den Generatorprozess eingegriffen werden, 28

COM Component Object Model. Proprietäre Technologie von Microsoft zur Realisierung von Komponenten nach dem Client/Server-Prinzip auf den Windows-Betriebssystemen. Wird in Zukunft vom verwalteten .Net-Komponentenmodell abgelöst, das auf der *Common Language Runtime*-Schicht (CLR) aufsetzt und damit eine höhere Abstraktion vom Betriebssystem erreicht. Die Interoperabilität zwischen den Modellen ist in beide Richtungen möglich, 10

ESS Embedded Systems and Sensors Engineering, eine Abteilung innerhalb des Forschungszentrums Informatik (FZI) Karlsruhe, die schwerpunktmäßig nach neuen Methoden in der ingenieursmäßigen Entwicklung eingebetteter Systeme und Sensoren forscht, 1

FLI , 5

FPGA Field Programmable Gate Array, ein frei programmierbarer Logikschaltkreis, 1, 5

FZI Das *Forschungszentrum Informatik* Karlsruhe ist eine gemeinnützige Forschungseinrichtung des Landes Baden-Württemberg, 1, 3, 41

GUI Graphical User Interface, grafische Benutzeroberfläche, 15, 33

IBM Die *International Business Machines Corporation* ist seit den 50er Jahren in der Hardware-, Software- und Betriebssystementwicklung tätig. Aus der Feder des Unternehmens stammt unter Anderem die Eclipse Plattform, 8

IDE Integrated Development Environment. Eine integrierte Entwicklungsmgebung fasst alle bei der Entwicklung anfallenden Aufgaben in einer (in der Regel grafischen, auf Fenstern basierenden) Oberfläche zusammen. Sie unterstützt den Programmierer bei seiner Tätigkeit. Beispiele für moderne Entwicklungsumgebungen sind die *Eclipse IDE*, *Sun NetBeans* oder *Microsoft Visual Studio*, 5, 6, 29, 45, 47, 48, 57, 70

JDT Java Development Tools. Bekannteste Erweiterung für Eclipse, sie macht aus der Rich-Client-Plattform eine Entwicklungsumgebung für Java, 10, 13, 14, 30, 50, 58

OSGi Open Services Gateway Initiative. Name eines Konsortiums und seiner entwickelten hardwareunabhängigen, generischen Dienstplattform basierend auf der Javamaschine. Die Plattform erlaubt dynamisches Hinzufügen und Entfernen von Dienstanwendungen (sog. *Bundles*) zur Laufzeit. Diensteffektspezifikationen der beteiligten Bundles sind feingranular auf Java-Ebene möglich, 10

PDE Plugin Development Environment. Der JDT zugehörige Erweiterung für die Eclipse-Plattform. Sie unterstützt die Entwicklung von Eclipse-Plugins, indem sie spezielle Editoren für die Schnittstellen zur Verfügung stellt. Andere wichtige Erweiterungen integrieren spezielle Konfigurationen zur Pluginausführung, einen Wizard zur Erstellung von Plugins aus Vorlagen und einen Wizard zum Exportieren von Pluginprojekten für die Weitergabe., 13

SoPC System on a Programmable Chip. Ein System, welches aus einem Prozessor samt anwendungsspezifischer Logik besteht und auf einem programmierbaren Logikbaustein (meist FPGA) untergebracht ist. Wird die Prozessorlogik ebenfalls anwendungsspezifisch programmiert, handelt es sich um ein SoPC mit *Softcore*-Prozessor., 5

SWT Standard Widget Toolkit, eine in Java implementierte Schnittstelle zur Verwendung der vom Betriebssystem bereitgestellten nativen Steuerelemente (GUI). Neben der verbesserten Performanz macht die Anwendung einen »gewohnteren Eindruck«. Nachteil gegenüber anderen Java-GUIs ist die erkaufte Plattformabhängigkeit. SWT ist jedoch für die meisten bekannten Betriebssysteme verfügbar, 72

Synthese Aufgabe der Synthese von technischen Systemen ist es, aus vorgegebenen bzw. geforderten Eigenschaften und Verhalten ein technisches System zu entwickeln, das eben diese Eigenschaften aufweist [Deu07b], 5

Workspace Der Workspace (Arbeitsbereich) bezeichnet in der Terminologie der Eclipse Plattform einen ausgewählten Ort im Dateisystem des Betriebsystems, an dem arbeitsplatzabhängige Einstellungen für die IDE sowie Projektdateien und deren Einstellungen gespeichert werden. Es muss immmer genau ein Arbeitsbereich geöffnet sein, 28, 33, 34, 47–57, 59, 60, 64

XSD XML Schema Definition, 85

Literatur

[Bal98] Balzert, Helmut: *Lehrbuch der Software-Technik*. 2. Heidelberg : Spektrum Akademischer Verlag, 1998. – ISBN 3–8274–0065–1

[BF93] Bibliographisches Institut (Hrsg.) ; F. A. Brockhaus AG (Hrsg.): *Duden Informatik*. 2. Auflage. 1993. – ISBN 3–4110–5232–5

[CR06] Clayberg, Eric ; Rubel, Dan: *Eclipse: Building Commercial-Quality Plug-ins (The Eclipse Series)*. 2. Addison-Wesley Professional, 2006. – ISBN 0–3214–2672–X

[Dau06a] Daum, Berthold: *Das Eclipse-Codebuch. 182 Tipps, Tricks und Lösungen für Eclipse-spezifische Probleme*. Dpunkt Verlag, 2006. – ISBN 3–8986–4376–X

[Dau06b] Daum, Berthold: *Java-Entwicklung mit Eclipse 3.2. Anwendungen, Plugins und Rich Clients*. Dpunkt Verlag, 2006. – ISBN 3–8986–4426–X

[Dau06c] Daum, Berthold: *Rich-Client-Entwicklung mit Eclipse 3.2. Anwendungen entwickeln mit der Rich Client Platform*. Dpunkt Verlag, 2006. – ISBN 3–8986–4427–8

[Deu07a] Deutsche Wikipedia: *Eclipse (IDE)*. http://de.wikipedia.org/w/index.php?title=Eclipse_%28IDE%29&oldid=27018052. Version: 29. Januar 2007

[Deu07b] Deutsche Wikipedia: *Synthese*. http://de.wikipedia.org/w/index.php?title=Synthese_%28Elektrotechnik%29&oldid=25527083. Version: 10. Januar 2007

[DFK+05] D'Anjou, Jim ; Fairbrother, Scott ; Kehn, Dan ; Kellerman, John ; McCarthy, Pat: *The Java Developer's Guide to Eclipse*. Addison-Wesley, München, 2005. – ISBN 3–8273–2254–5

[Ecl07] Eclipse Foundation: *Eclipse Project*. http://www.eclipse.org/. Version: 29. Januar 2007

[GB05] Gamma, Erich ; Beck, Kent: *Eclipse erweitern. Prinzipien, Patterns und Plug-Ins*. Addison-Wesley, München, 2005. – ISBN 3–8273–2238–3

[GHJV96] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. 1. Bonn : Addison-Wesley, 1996. – ISBN 3–89319–950–0

[JSY03] Johnson, Brian ; Skibo, Craig ; Young, Marc: *Inside Microsoft® Visual Studio® .NET. Effizientes Programmieren mit dem Visual Studio .NET 2003*. 1. Microsoft Press Deutschland, 2003. – ISBN 3–8606–3683–9

[LK07] Lutz, Benjamin ; Köllner, Christian: An Integrated Design Environment for Embedded HW/SW Systems. In: *Proceedings of the Embedded World, Nuremberg, Germany* (2007), February

[MG97] Micheli, Giovanni D. ; Gupta, Rajesh K.: Hardware/Software Co-Design. In: *Proceedings of the IEEE* 85 (1997), 3, S. 349–365

[Mic07a] Microsoft Corporation: *Microsoft Visual Studio Extensibility*. http://www.devx.com/vstudioextensibility/. Version: 29. Januar 2007

[Mic07b] Microsoft Corporation: *Visual Studio 2005 Extensibility Center*. http://msdn.microsoft.com/vstudio/extend/. Version: 29. Januar 2007

[PHH06] Peh, Diana ; Hannemann, Alethea ; Hague, Nola: *BIRT: A Field Guide to Reporting (The Eclipse Series)*. 1. Addison-Wesley Professional, 2006. – ISBN 0–3214–4259–8

[Sie02] Siedersleben, Johannes: *Softwaretechnik. Praxiswissen für Softwareingenieure*. 2. Hanser Fachbuchverlag, 2002. – ISBN 3446218432

[SVC07] SVC Delivery: *Eclipse XML Author v0.9.12*. http://www.svcdelivery.com/xmlauthor/. Version: 12. April 2007

[T+07] Tadashi, Kobayashi u. a.: *Eclipse VHDL/Verilog Editor v0.5.1*. http://veditor.sourceforge.net/. Version: 12. April 2007

[TRI07] TRIAS Mikroelektronik GmbH: *Impulse CoDeveloper Universal*. http://www.trias-mikro.de/pdfs/eda/FPGA/TRIAS_CoDeveloper_Universal.pdf. Version: 20. März 2007